Computational Complexity:

A Conceptual Perspective

Oded Goldreich

Department of Computer Science and Applied Mathematics
Weizmann Institute of Science, Rehovot, ISRAEL.

July 21, 2006

Chapter 6

Randomness and Counting

I owe this almost atrocious variety to an institution which other republics do not know or which operates in them in an imperfect and secret manner: the lottery.

Jorge Luis Borges, The Lottery In Babylon

So far, 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).

The quantitative nature of the failure probability of probabilistic algorithm provides one connection between probabilistic algorithms and counting problems. The latter are indeed a new type of computational problems, and our focus is on counting efficiently recognizable objects (e.g., NP-witnesses for a given instance of set in $\mathcal{NP}$). Randomized procedures turn out to play an important role in the study of such counting problems.

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

We then turn to counting problems; specifically, counting the number of solutions for an instance of a search problem in $\mathcal{PC}$ (or, equivalently, counting the number of NP-witnesses for an instance of a decision problem in $\mathcal{NP}$). We distinguish between exact counting and approximate counting (in the sense of relative approximation). In particular, while

any problem in $\mathcal{PH}$ is reducible to the exact counting class $\#\mathcal{P}$, approximate counting (for $\#\mathcal{P}$) is (probabilisticly) reducible to $\mathcal{NP}$.

Additional related topics include the $\#\mathcal{P}$-completeness of various counting problems (e.g., counting the number of satisfying assignments to a given CNF formula and counting the number of perfect matchings in a given graph), the complexity of searching for unique solutions, and the relation between approximate counting and generating (almost) uniformly distributed solutions.

Prerequisites: We assume basic familiarity with elementary probability theory (see Section D.1). In Section 6.2 we will rely extensively on the formulation presented in Section 2.1 (i.e., the "NP search problem" class $\mathcal{PC}$ as well as the sets $R(x) \stackrel{\rm def}{=} \{y : (x,y) \in R\}$, and $S_R \stackrel{\rm def}{=} \{X : R(x) \neq \emptyset\}$ defined for every $R \in \mathcal{PC}$).

6.1 Probabilistic Polynomial-Time

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 (see the first item of Definition 2.7), *but the definition of its computation is fundamentally different*. Specifically, whereas Definition 2.7 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 in the corresponding probabilistic computation one of these triples is selected at random (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; see Exercise 2.4.)

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 real random 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: Clearly, the aforementioned residual deterministic machine M' yields the on-line machine M that on input x selects at random a string r of adequate length, and invokes $M'(x,r)$. On the other hand, the computation of any on-line machine M is captured by the residual machine M' that emulates the actions of $M(x)$ based on an auxiliary input r (obtained by M' and representing a possible outcome of the internal coin tosses of M). (Indeed, there is no harm in supplying more coin tosses than are actually used by M, and so the length of the aforementioned auxiliary input may be set to equal the time complexity of M.) For sake of clarity and future reference, we state the following definition.

Definition 6.1 (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 6.1). 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 subsections, 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-reductions), 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 Definition 2.9, 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 Definition 2.10, 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 $\mathsf{Pr}[\chi_{S'}(A(x))=\chi_S(x)] \geq 1-\mu(|x|)$, 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 6.2.

6.1.1 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 (see Definition 1.1). 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 $\mathsf{Pr}[A(x) \in R(x)] > 1-\mu(|x|)$ and for every $x \notin S_R$ it holds that $\mathsf{Pr}[A(x) = \perp] > 1-\mu(|x|)$, 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. Indeed, 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 and quantitative relaxations of it allow for error-reduction (see Exercise 6.3).

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 6.2 (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 6.4).

6.1.1.1 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 (see, e.g., Chapter 9, Section 10.1.2, and Appendix C) and seem essential in others (see, e.g., Sections 6.2.2-6.2.4). 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 $\mathcal{BPP} = \mathcal{P}$ (see Part 1 of Theorem 8.19). 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 $\mathrm{poly}(t(\cdot))$. Furthermore, for some concrete problems (most notably primality testing (cf. §6.1.1.2)), 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]

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 6.11). 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 6.7.

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 Section 8.4 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 Chapter 8.

[1] Such a result would address a fundamental question regarding the power of randomness. By analogy, Theorem 9.4 establishing that $\mathcal{IP} = \mathcal{PSPACE}$ does not diminish the importance of any of these classes.

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

Proof: Recall that $\mathcal{P}$/poly contains undecidable problems (Theorem 3.7), 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 $\Pr[A(x, U_{p(|x|)}) \neq \chi_S(x)] < 2^{-|x|}$. Using a union bound, it follows that $\Pr_{r\in\{0,1\}^{p(n)}}[\exists x \in \{0,1\}^n \text{ s.t. } A(x,r) \neq \chi_S(x)] < 1$. 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). ■

Digest. The proof of Theorem 6.3 combines error-reduction with a simple application of the Probabilistic Method (cf. [10]), where the latter refers to proving the existence of an object by analyzing the probability that a random object is adequate. In this case, we sought an non-uniform advice, and proved it existence by analyzing the probability that a random advice is good. The latter event was analyzed by identifying the space of advice with the set of possible sequences of internal coin tosses of a randomized algorithm.

6.1.1.2 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]

[2] In other contexts (see, e.g., Chapters 7 and 8), 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 $x^2 \equiv r^2 \pmod{p}$ has two solutions modulo p (i.e., r and $p - r$).

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.

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; see Section 2.4.1).

Construction 6.4 (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 $r' \leftarrow$ `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 6.14).

Proposition 6.5 *Construction 6.4 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 6.5 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 6.5 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 6.4 always accepts (because in this case, for every $r \in \{1, ..., N-1\}$, it holds that `sqrt`$(r^2 \bmod N, N) \in \{r, N-r\}$). 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

[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.

$\texttt{sqrt}(s, N)$ 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 6.4 illustrates an interesting aspect of randomized algorithms (or rather reductions); that is, the ability to hide information from a subroutine. Specifically, Construction 6.4 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).

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).

6.1.2 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 6.1.1).

Definition 6.6 (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* $\Pr[A(x)=1] \geq 1/2$ *and for every $x \notin S$ it holds that* $\Pr[A(x)=0] = 1$.

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 6.5).

Observe that $\mathcal{RP} \subseteq \mathcal{NP}$ (see Exercise 6.11) and that $\mathcal{RP} \subseteq \mathcal{BPP}$ (by the aforementioned error-reduction). Defining $\text{co}\mathcal{RP} = \{\{0,1\}^* \setminus S : S \in \mathcal{RP}\}$, note that co$\mathcal{RP}$ corresponds to the opposite direction of one-sided error probability. That is, *a decision problem S is in* co$\mathcal{RP}$ *if there exists a probabilistic polynomial-time algorithm A such that for every $x \in S$ it holds that* $\Pr[A(x)=1] = 1$ *and for every $x \notin S$ it holds that* $\Pr[A(x)=0] \geq 1/2$.

[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 that is contained in $\mathcal{NP}$.

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 co$\mathcal{RP}$ by *one-sided error* randomized Karp-reductions, where the actual statement refers to the promise problem versions of both classes (briefly defined in the following paragraph). Note that $\mathcal{BPP}$ is trivially reducible to co$\mathcal{RP}$ by *two-sided error* randomized Karp-reductions whereas a deterministic reduction of $\mathcal{BPP}$ to co$\mathcal{RP}$ would imply $\mathcal{BPP} = \text{co}\mathcal{RP} = \mathcal{RP}$ (see Exercise 6.8).

First, we refer the reader to the general discussion of promise problems in Section 2.4.1. Analogously to Definition 2.30, we say that the promise problem $\Pi = (S_{\text{yes}}, S_{\text{no}})$ is in (the promise problem extension of) $\mathcal{BPP}$ *if there exists a probabilistic polynomial-time algorithm A such that for every $x \in S_{\text{yes}}$ it holds that* $\Pr[A(x)=1] \geq 2/3$ *and for every $x \in S_{\text{no}}$ it holds that* $\Pr[A(x)=0] \geq 2/3$. Similarly, Π is in co$\mathcal{RP}$ if for every $x \in S_{\text{yes}}$ it holds that $\Pr[A(x) = 1] = 1$ and for every $x \in S_{\text{no}}$ it holds that $\Pr[A(x)=0] \geq 1/2$. Probabilistic reductions among promise problems are defined by adapting the conventions of Section 2.4.1; specifically, queries that violate the promise at the target of the reduction may be answered arbitrarily.

Theorem 6.7 *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, using the conventions of Section 3.2.2 and 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 6.7 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 6.7).

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 6.7. 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 6.7 (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 $\Pr_{r\in\{0,1\}^{2p'(|x|)}}[A'(x,r)\neq\chi_S(x)] < 2^{-(p'(|x|)+1)}$; 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 6.9). Anyhow, such a small error probability allows a partition of the string r such that one part accounts for the entire error probability on yes-instances while the other part accounts for the error probability on no-instances. Specifically, for every $x \in S$, it holds that $\Pr_{r'\in\{0,1\}^{p'(|x|)}}[(\forall r'' \in \{0,1\}^{p'(|x|)})\, A'(x, r'r'') = 1] > 1/2$, whereas for every $x \notin S$ and every $r' \in \{0,1\}^{p'(|x|)}$ it holds that $\Pr_{r''\in\{0,1\}^{p'(|x|)}}[A'(x,r'r'') = 1] < 1/2$. Thus, 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 6.10. 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 $\Pr[A(x)\neq\chi(x)] < \mu(|x|)$, 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

$$\Pr_{r\in\{0,1\}^{p(|x|)}}[A'(x,r)\neq\chi(x)] \;<\; \mu(|x|) \;<\; \frac{1}{2p(|x|)} \tag{6.1}$$

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. (6.1) 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 $s_1, ..., s_m \in \{0,1\}^m$, for at least half of the possible $r \in \{0,1\}^m$ there exists an i such that $A'(x, r \oplus s_i) = 1$ holds. 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 $\Pi_{\rm no}$).

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:

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

where the last inequality is due to Eq. (6.1). It follows that if $\chi(x) = 1$ then $\mathsf{Pr}[M(x) \in \Pi_{\rm yes}] \gg 1/2$. 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. ■

BPP is in PH. The traditional presentation of the ideas underlying the proof of Theorem 6.7 uses them for showing that *$\mathcal{BPP}$ is in the Polynomial-time Hierarchy* (where both classes refer to standard decision problems). Specifically, to prove that $\mathcal{BPP} \subseteq \Sigma_2$ (see Definition 3.8), define the polynomial-time computable predicate $\varphi(x, \overline{s}, r) \stackrel{\rm def}{=} \bigvee_{i=1}^m (A'(x, s_i \oplus r) = 1)$, and observe that

$$\chi(x) = 1 \quad \Rightarrow \quad \exists \overline{s}\, \forall r \ \ \varphi(x, \overline{s}, r) \tag{6.2}$$

$$\chi(x) = 0 \quad \Rightarrow \quad \forall \overline{s}\, \exists r \ \ \neg\varphi(x, \overline{s}, r) \tag{6.3}$$

(where Eq. (6.3) is equivalent to $\neg\exists \overline{s}\, \forall r \ \varphi(x, \overline{s}, r)$). Note that Claim 1 (in the proof of Theorem 6.7) establishes that *most* sequences $\overline{s}$ satisfy $\forall r\, \varphi(x, \overline{s}, r)$, whereas Eq. (6.2) 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. (6.3) 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 $\mathcal{NP}$ defined in Section 9.1).[6]

[6] Specifically, the class $\mathcal{MA}$ is defined by allowing the verification algorithm V in Definition 2.5 to be probabilistic and err on no-instances; that is, for every $x \in S$ there exists $y \in \{0,1\}^{\mathrm{poly}(|x|)}$ 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. (6.2) and Claim 2. Other randomized versions of $\mathcal{NP}$ (i.e., variants of $\mathcal{MA}$) are considered in Exercise 6.12.

6.1.3 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 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* $\mathsf{Pr}[A(x) \in \{\chi_S(x), \perp\}] = 1$ *and* $\mathsf{Pr}[A(x) = \chi_S(x)] \geq 1/2$*, 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 6.6).

Theorem 6.8 $\mathcal{ZPP} = \mathcal{RP} \cap \text{co}\mathcal{RP}$.

Proof Sketch: The fact that $\mathcal{ZPP} \subseteq \mathcal{RP}$ (as well as $\mathcal{ZPP} \subseteq \text{co}\mathcal{RP}$) 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 $\mathcal{RP} \cap \text{co}\mathcal{RP} \subseteq \mathcal{ZPP}$ we combine the two algorithm guaranteed for a set in $\mathcal{RP} \cap \text{co}\mathcal{RP}$. 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 $\perp$. □

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 6.7).

6.1.4 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.

6.1.4.1 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 (see Section 5.3). Specifically, the on-line and the off-line versions (formulated in Definition 6.1) 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 outcome 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 Section 5.3), 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 6.15, 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. Modulo this convention, the class $\mathcal{RL}$ (resp., $\mathcal{BPL}$) relates to $\mathcal{NL}$ analogously to the relation of $\mathcal{RP}$ (resp., $\mathcal{BPP}$) to $\mathcal{NP}$. Specificially, the probabilistic acceptance condition of $\mathcal{RL}$ (resp., $\mathcal{BPL}$) is as in the case of $\mathcal{RP}$ (resp., $\mathcal{BPP}$).

Definition 6.9 (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).

6.1.4.2 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 in Section 5.2.4 we proved 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 6.10 *On input* (G, s, t)*, the randomized algorithm starts a* $\text{poly}(|G|)$*-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 6.11 implies that undirected connectivity is indeed in $\mathcal{RL}$.

Proposition 6.11 *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 6.16. Next, we let $\text{cover}(G)$ 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 $\text{cover}(G)$. Towards this end, we consider an arbitrary directed cyclic-tour

C that visits all vertices in G, and note that

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

In particular, selecting C as a traversal of some spanning tree of G, we conclude that $\operatorname{cover}(G) < 4 \cdot |V| \cdot |E|$. 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. □

6.2 Counting

We now turn to a new type of computational problems, which vastly generalize decision problems of the NP-type. We refer to counting problems, and more specifically to counting objects that can be efficiently recognized. The search and decision versions of NP provide suitable definitions of efficiently recognized objects, which in turn yield corresponding counting problems:

1. For each search problem having efficiently checkable solutions (i.e., a relation $R \subseteq \{0,1\}^* \times \{0,1\}^*$ in $\mathcal{PC}$ (see Definition 2.3)), we consider the problem of counting the number of solutions for a given instance. That is, on input x, we are required to output $|\{y : (x,y) \in R\}|$.
2. For each decision problem S in $\mathcal{NP}$, and each corresponding verification procedure V (as in Definition 2.5), we consider the problem of counting the number of NP-witnesses for a given instance. That is, on input x, we are required to output $|\{y : V(x,y) = 1\}|$.

We shall consider these types of counting problems as well as relaxations (of these counting problems) that refer to approximating the said quantities (see Sections 6.2.1 and 6.2.2, respectively). Other related topics include "problems with unique solutions" (see Section 6.2.3) and "uniform generation of solutions" (see Section 6.2.4). Interestingly, randomized procedures will play an important role in the results regarding the aforementioned types of problems.

6.2.1 Exact Counting

In continuation to the foregoing discussion, we define the class of problems concerned with counting efficiently recognized objects. (Recall that $\mathcal{PC}$ denotes the class of search problems having polynomially long solutions that are efficiently checkable; see Definition 2.3.)

Definition 6.12 (counting efficiently recognized objects – $\#\mathcal{P}$): *The class $\#\mathcal{P}$ consists of all functions that count solutions to a search problem in $\mathcal{PC}$. That is, $f : \{0,1\}^* \to \mathbb{N}$ is in $\#\mathcal{P}$ if there exists $R \in \mathcal{PC}$ such that, for every x, it holds that $f(x) = |R(x)|$, where $R(x) = \{y : (x,y) \in R\}$. In this case we say that f is the* counting problem associated with *R, and denote the latter by $\#R$* (i.e., $\#R = f$).

Every decision problem in $\mathcal{NP}$ is Cook-reducible to $\#\mathcal{P}$, because every such problem can be cast as deciding membership in $S_R = \{x : |R(x)| > 0\}$ for some $R \in \mathcal{PC}$ (see Section 2.1.2). It also holds that $\mathcal{BPP}$ is Cook-reducible to $\#\mathcal{P}$ (see Exercise 6.17). The class $\#\mathcal{P}$ is sometimes defined in terms of decision problems, as is implicit in the following proposition.

Proposition 6.13 (a decisional version of $\#\mathcal{P}$): *For any $f \in \#\mathcal{P}$, deciding membership in $S_f \stackrel{\text{def}}{=} \{(x,N) : f(x) \geq N\}$ is computationally equivalent to computing f.*

Actually, the claim holds for any function $f : \{0,1\}^* \to \mathbb{N}$ for which there exists a polynomial p such that for every $x \in \{0,1\}^*$ it holds that $f(x) \leq 2^{p(|x|)}$.

Proof: Since the relation R vouching for $f \in \#\mathcal{P}$ (i.e., $f(x) = |R(x)|$) is polynomially bounded, there exists a polynomial p such that for every x it holds that $f(x) \leq 2^{p(|x|)}$. Deciding membership in S_f is easily reduced to computing f (i.e., we accept the input (x,N) if and only if $f(x) \geq N$). Computing f is reducible to deciding S_f by using a binary search (see Exercise 2.9). This relies on the fact that, on input x and oracle access to S_f, we can determine whether or not $f(x) \geq N$ by making the query (x,N). Note that we know a priori that $f(x) \in [0, 2^{p(|x|)}]$. ■

The counting class $\#\mathcal{P}$ is also related to the problem of enumerating all possible solutions to a given instance (see Exercise 6.19).

6.2.1.1 On the power of $\#\mathcal{P}$

As indicated, $\mathcal{NP} \cup \mathcal{BPP}$ is (easily) reducible to $\#\mathcal{P}$. Furthermore, as stated in Theorem 6.14, the entire Polynomial-Time Hierarchy (as defined in Section 3.2) is Cook-reducible to $\#\mathcal{P}$ (i.e., $\mathcal{PH} \subseteq \mathcal{P}^{\#\mathcal{P}}$). On the other hand, any problem in $\#\mathcal{P}$ is solvable in polynomial space, and so $\mathcal{P}^{\#\mathcal{P}} \subseteq \mathcal{PSPACE}$.

Theorem 6.14 *Every set in $\mathcal{PH}$ is Cook-reducible to $\#\mathcal{P}$.*

We do not present a proof of Theorem 6.14 here, because the known proofs are rather technical. Furthermore, one main idea underlying these proofs appears in a more clear form in the proof of Theorem 6.27. Nevertheless, in Section F.1 we present a proof of a related result, which implies that $\mathcal{PH}$ is reducible to $\#\mathcal{P}$ via *randomized* Karp-reductions.

6.2.1.2 Completeness in $\#\mathcal{P}$

The definition of $\#\mathcal{P}$-completeness is analogous to the definition of $\mathcal{NP}$-completeness. That is, *a counting problem f is $\#\mathcal{P}$-complete if $f \in \#\mathcal{P}$ and every problem in $\#\mathcal{P}$ is Cook-reducible to f.*

We claim that the counting problems associated with the NP-complete problems presented in Section 2.3.3 are all $\#\mathcal{P}$-complete. We warn that this fact is not due to the mere NP-completeness of these problems, but rather to an additional property of the reductions establishing their NP-completeness. Specifically, the Karp-reductions that were used (or variants of them) have the extra property of preserving the number of NP-witnesses (as captured by the following definition).

Definition 6.15 (parsimonious reductions): *Let $R, R' \in \mathcal{PC}$ and let g be a Karp-reduction of $S_R = \{x : R(x) \neq \emptyset\}$ to $S_{R'} = \{x : R'(x) \neq \emptyset\}$, where $R(x) = \{y : (x,y) \in R\}$ and $R'(x) = \{y : (x,y) \in R'\}$. We say that g is* parsimonious (with respect to R and R') *if for every x it holds that $|R(x)| = |R'(g(x))|$. In such a case we say that g is a* parsimonious reduction of R to R'.

We stress that the condition of being parsimonious refers to the two underlying relations R and R' (and not merely to the sets S_R and $S_{R'}$). The requirement that g is a Karp-reduction is partially redundant, because if g is polynomial-time computable and for every x it holds that $|R(x)| = |R'(g(x))|$, then g constitutes a Karp-reduction of S_R to $S_{R'}$. Specifically, $|R(x)| = |R'(g(x))|$ implies that $|R(x)| > 0$ (i.e., $x \in S_R$) if and only if $|R'(g(x))| > 0$ (i.e., $g(x) \in S_{R'}$). The reader may easily verify that the Karp-reduction underlying the proof of Theorem 2.18 as well as many of the reductions used in Section 2.3.3 are parsimonious (see Exercise 2.17).

Theorem 6.16 *Let $R \in \mathcal{PC}$ and suppose that every search problem in $\mathcal{PC}$ is parsimoniously reducible to R. Then the counting problem associated with R is $\#\mathcal{P}$-complete.*

Proof: Clearly, the counting problem associated with R, denoted $\#R$, is in $\#\mathcal{P}$. To show that every $f' \in \#\mathcal{P}$ is reducible to f, we consider the relation $R' \in \mathcal{PC}$ that is counted by f'; that is, $\#R' = f'$. Then, by the hypothesis, there exists a parsimonious reduction g of R' to R. This reduction also reduces $\#R'$ to $\#R$; specifically, $\#R'(x) = \#R(g(x))$ for every x. ∎

Corollaries. As an immediate corollary of Theorem 6.16, we get that counting the number of satisfying assignments to a given CNF formula is $\#\mathcal{P}$-complete. Similar statement hold for all the other NP-complete problems mentioned in Section 2.3.3 and in fact for all NP-complete problems listed in [81]. These corollaries follow from the fact that all known reductions among natural NP-complete problems are either parsimonious or can be easily modified to be so.

We conclude that many counting problems associated with NP-complete search problems are $\#\mathcal{P}$-complete. It turns out that also counting problems associated with efficiently solvable search problems may be $\#\mathcal{P}$-complete.

Theorem 6.17 *There exist $\#\mathcal{P}$-complete counting problems that are associated with efficiently solvable search problems. That is, there exists $R \in \mathcal{PF}$* (see Definition 2.2) *such that $\#R$ is $\#\mathcal{P}$-complete.*

Proof: Consider the relation $R_{\mathtt{dnf}}$ consisting of pairs (ϕ, τ) such that ϕ is a DNF formula and τ is an assignment satisfying it. Note that the search problem of $R_{\mathtt{dnf}}$ is easy to solve (e.g., by picking an arbitrary truth assignment that satisfies the first term in the input formula). To see that $\#R_{\mathtt{dnf}}$ is $\#\mathcal{P}$-complete consider the following reduction from $\#R_{\mathtt{SAT}}$ (which is $\#\mathcal{P}$-complete by Theorem 6.16). Given a CNF formula ϕ, transform $\neg\phi$ into a DNF formula ϕ' by applying de-Morgan's Law, and return $2^n - \#R_{\mathtt{dnf}}(\phi')$, where n denotes the number of variables in ϕ (resp., ϕ'). ∎

Reflections. We note that Theorem 6.17 is not established by a parsimonious reduction (and refer the reader to more artifical $\#\mathcal{P}$-complete problems presented in Exercise 6.18). This fact should not come as a surprise because a parsimonious reduction of $\#R'$ to $\#R$ implies that $S_{R'} = \{x : \exists y \text{ s.t. } (x,y) \in R'\}$ is reducible to $S_R = \{x : \exists y \text{ s.t. } (x,y) \in R\}$, where in our case $S_{R'}$ is NP-Complete while $S_R \in \mathcal{P}$ (since $R \in \mathcal{PF}$). Nevertheless, the proof of Theorem 6.17 is related to the hardness of some underlying decision problem (i.e., the problem of deciding whether a given DNF formula is a tautology (i.e., whether $\#R_{\mathtt{dnf}}(\phi') = 2^n$)). But does there exist a $\#\mathcal{P}$-complete problem that is "not based on some underlying NP-complete decision problem"? Amazingly enough, the answer is positive.

Theorem 6.18 *Counting the number of perfect matchings in a bipartite graph is $\#\mathcal{P}$-complete.*[7]

Equivalently (see Exercise 6.20), the problem of computing the permanent of matrices with 0/1-entries is $\#\mathcal{P}$-complete. Recall that the permanent of an n-by-n matrix $M = (m_{i,j})$, denoted $\mathtt{perm}(M)$, equals the sum over all permutations π of $[n]$ of the products $\prod_{i=1}^{n} m_{i,\pi(i)}$. Theorem 6.18 is proven by composing the following two (many-to-one) reductions (asserted in Propositions 6.19 and 6.20, respectively) and using the fact that $\#R_{\mathtt{3SAT}}$ is $\#\mathcal{P}$-complete (see Theorem 6.16 and Exercise 2.17). Needless to say, the resulting reduction is not parsimonious.

Proposition 6.19 *The counting problem of* 3SAT (i.e., $\#R_{\mathtt{3SAT}}$) *is reducible to computing the permanent of integer matrices. Furthermore, there exists an* even *integer $c > 0$ and a finite set of integers I such that, on input a 3CNF formula ϕ, the reduction produces an integer matrix M_ϕ with entries in I such that* $\mathtt{perm}(M_\phi) = c^m \cdot \#R_{\mathtt{3SAT}}(\phi)$ *where m denotes the number of clauses in ϕ.*

The original proof of Proposition 6.19 uses $c = 2^{10}$ and $I = \{-1, 0, 1, 2, 3\}$. It can be shown (see Exercise 6.21 (which relies on Theorem 6.27)) that, for every integer $n > 1$ that is relatively prime to c, computing the permanent modulo n is NP-hard (under randomized reductions). Thus, using the case of $c = 2^{10}$, this means that computing the permanent modulo n is NP-hard for any odd $n > 1$. In contrast, computing the permanent modulo 2 (which is equivalent to computing the determinant modulo 2) is easy (i.e., can be done in polynomial-time and even in $\mathcal{NC}$). Thus, assuming $\mathcal{NP} \not\subseteq \mathcal{BPP}$, Proposition 6.19 cannot hold for an odd c (because by Exercise 6.21 it would follow that computing the permanent modulo 2 is NP-Hard). We also note that, assuming $\mathcal{P} \neq \mathcal{NP}$, Proposition 6.19 cannot possibly hold for a set I containing only non-negative integers (see Exercise 6.22).

Proposition 6.20 *Computing the permanent of integer matrices is reducible to computing the permanent of 0/1-matrices. Furthermore, the reduction maps any integer matrix A into a 0/1-matrix A'' such that the permanent of A can be easily computed from A and the permanent of A''.*

[7] See Section G.1 for basic terminology regarding graphs.

Teaching note: We do not recommend presenting the proofs of Propositions 6.19 and 6.20 in class. The high-level structure of the proof of Proposition 6.19 has the flavor of some sophisticated reductions among NP-problems, but the crucial point is the existence of adequate gadgets. We do not know of a high-level argument establishing the existence of such gadgets nor of any intuition as to why such gadgets exist.[8] Instead, the existence of such gadgets is proved by a design that is both highly non-trivial and *ad hoc* in nature. Thus, the proof of Proposition 6.19 boils down to a complicated design problem that is solved in a way that has little pedagogical value. In contrast, the proof of Proposition 6.20 uses two simple ideas that can be useful in other settings. With suitable hints, this proof can be used as a good exercise.

Proof of Proposition 6.19: We will use the correspondence between the permanent of a matrix A and the sum of the weights of the cycle covers of the weighted directed graph represented by the matrix A. A cycle cover of a graph is a collection of simple[9] *vertex-disjoint* directed cycles that covers all the graph's vertices, and its weight is the product of the weights of the corresponding edges. The SWCC of a weighted directed graph is the sum of the weights of all its cycle covers.

Given a 3CNF formula ϕ, we construct a directed weighted graph G_ϕ such that the SWCC of G_ϕ equals equals $c^m \cdot \#R_{\mathtt{3SAT}}(\phi)$, where c is a universal constant and m denotes the number of clauses in ϕ. We may assume, without loss of generality, that each clause of ϕ has exactly three literals (which are not necessarily distinct).

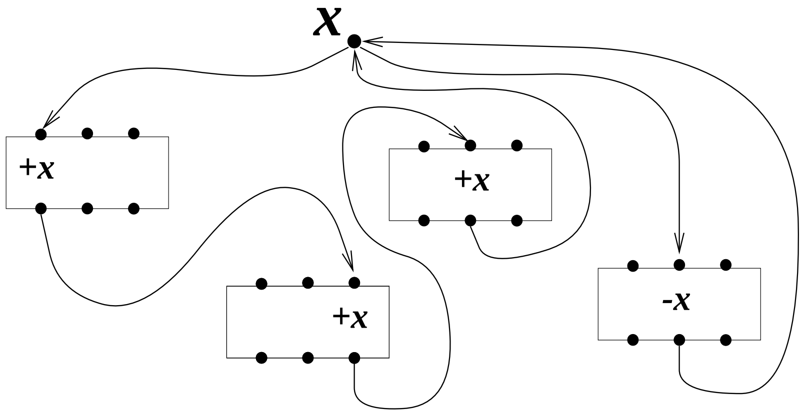


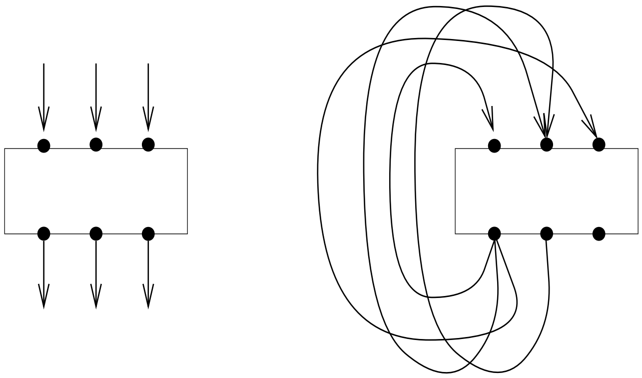
Figure 6.1: Tracks connecting gadgets for the reduction to cycle cover.

We start with a high-level description (of the construction) that refers to (clause) gadgets, each containing some internal vertices and internal (weighted) edges, which are *unspecified at this point.* In addition, each gadget has three pairs of designated vertices, one pair per each literal appearing in the clause, where one vertex in the

[8] Indeed, the conjecture that such gadgets exist can only be attributed to ingenuity.

[9] Here a simple cycle is a strongly connected directed graph in which each vertex has a single incoming (resp., outgoing) edge. In particular, self-loops are allowed.

pair is designated as an entry vertex and the other as an exit vertex. The graph G_ϕ consists of m such gadgets, one per each clause (of ϕ), and n auxiliary vertices, one per each variable (of ϕ), as well as some *additional directed edges*, each having weight 1. Specifically, for each variable, we introduce two tracks, one per each of the possible literals of this variable. The track associated with a literal consists of directed edges (each having weight 1) that form a simple "cycle" passing through the corresponding (auxiliary) vertex as well as through the designated vertices that correspond to the occurrences of this literal in the various clauses. Specifically, for each such occurrence, the track enters the corresponding clause gadget at the entry-vertex corresponding to this literal and exits at the corresponding exit-vertex. (If a literal does not appear in ϕ then the corresponding track is a self-loop on the corresponding variable.) See Figure 6.1 showing the two tracks of a variable x that occurs positively in three clauses and negatively in one clause. The entry-vertices (resp., exit-vertices) are drawn on the top (resp., bottom) part of each gadget.



On the left is a gadget with the track edges adjacent to it (as in the real construction). On the right is a gadget and four out of the nine external edges (two of which are nice) used in the analysis.

Figure 6.2: External edges for the analysis of the clause gadget

For the purpose of stating the desired properties of the clause gadget, we augment the gadget by nine external edges (of weight 1), one per each pair of (not necessarily matching) entry and exit vertices such that the edge goes from the exit-vertex to the entry-vertex (see Figure 6.2). (We stress that this is an auxiliary construction that differs from and yet is related to the use of gadgets in the foregoing construction of G_ϕ.) The three edges that link the designated pairs of vertices that correspond to the three literals are called nice. We say that a collection of edges C (e.g., a collection of cycles) uses the external edges S if the intersection of C with the set of the (nine) external edges equals S. We postulate the following three properties of the clause gadget.

1. The sum of the weights of all cycle covers (of the gadget) that do not use any external edge (i.e., use the empty set of external edges) equals zero.

2. Let $V(S)$ denote the set of vertices incident to S, and say that S is nice if it is non-empty and the vertices in $V(S)$ can be perfectly matched using nice edges.[10] Then, there exists a constant c (indeed the one postulated in the proposition's claim) such that, for any nice set S, the sum of the weights of all cycle covers that use the external edges S equals c.

3. For any non-nice set S of external edges, the sum of the weights of all cycle covers that use the external edges S equals zero.

Note that the foregoing three cases exhaust all the possible ones, and that the set of external edges used by a cycle cover must be a matching (i.e., these edges are vertex disjoint). Using the foregoing conditions, it can be shown that each satisfying assignment of ϕ contributes exactly c^m to the SWCC of G_ϕ (see Exercise 6.23). It follows that the SWCC of G_ϕ equals $c^m \cdot \#R_{\mathtt{3SAT}}(\phi)$.

Having established the validity of the abstract reduction, we turn to the implementation of the clause gadget. The first implementation is a *Deus ex Machina*, with a corresponding adjacency matrix depicted in Figure 6.3. Its validity (for the value $c = 12$) can be verified by computing the permanent of the corresponding sub-matrices (see analogous analysis in Exercise 6.25).

A more structured implementation of the clause gadget is depicted in Figure 6.4, which refers to a (hexagon) box to be implemented later. The box contains several vertices and weighted edges, but only two of these vertices, called terminals, are connected to the outside (and are shown in Figure 6.4). The clause gadget consists of five copies of this box, where three copies are designated for the three literals of the clause (and are marked `LB1`, `LB2`, and `LB3`), as well as additional vertices and edges shown in Figure 6.4. In particular, the clause gadget contains the six aforementioned designated vertices (i.e., a pair of entry and exit vertices per each literal), two additional vertices (shown at the two extremes of the figure), and some edges (all having weight 1). Each designated vertex has a self-loop, and is incident to a single additional edge that is outgoing (resp., incoming) in case the vertex is an entry-vertex (resp., exit-vertex) of the gadget. The two terminals of each box that is associated with some literal are connected to the corresponding pair of designated vertices (e.g., the outgoing edge of `entry1` is incident at the right terminal of the box `LB1`). Note that the five boxes reside on a directed path (going from left to right), and the only edges going in the opposite direction are those drawn below this path.

In continuation to the foregoing, we wish to state the desired properties of the box. Again, we do so by considering the augmentation of the box by external edges (of weight 1) incident at the specified vertices. In this case (see Figure 6.5), we have a pair of anti-parallel edges connecting the two terminals of the box as well as two self-loops (one on each terminal). We postulate the following three properties of the box.

[10]Clearly, any non-empty set of nice edges is a nice set. Thus, a singleton set is nice if and only if the corresponding edge is nice. On the other hand, any set S of three (vertex-disjoint) external edges is nice, because $V(S)$ has a perfect matching using all three nice edges. Thus, the notion of nice sets is "non-trivial" only for sets of two edges. Such a set S is nice if and only if $V(S)$ consists of two pairs of corresponding designated vertices.

The gadget uses eight vertices, where the first six are the designated (entry and exit) vertices. The entry-vertex (resp., exit-vertex) associated with the i^{th} literal is numbered i (resp., $i+3$). The corresponding adjacency matrix follows.

$$\begin{pmatrix} 1 & 0 & 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 1 & -1 & 0 & 1 & 1 \\ 0 & 0 & -1 & -1 & 2 & 0 & 1 & 1 \\ 0 & 0 & 0 & -1 & -1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 2 & -1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}$$

Note that the edge $3 \to 6$ can be contracted, but the resulting 7-vertex graph will not be consistent with our (inessentially stringent) definition of a gadget by which the six designated vertices should be distinct.

Figure 6.3: A Deus ex Machina clause gadget for the reduction to cycle cover.

1. The sum of the weights of all cycle covers (of the box) that do not use any external edge equals zero.

2. There exists a constant b (in our case $b = 4$) such that, for each of the two anti-parallel edges, the sum of the weights of all cycle covers that use this edge equals b.

3. For any (non-empty) set S of the self-loops, the sum of the weights of all cycle covers (of the box) that use S equals zero.

Note that the foregoing three cases exhaust all the possible ones. It can be shown that the conditions regarding the box imply that the construction presented in Figure 6.4 satisfies the conditions that were postulated for the clause gadget (see Exercise 6.24). Specifically, we have $c = b^5$. As for box itself, a smaller *Deus ex Machina* is provided by the following 4-by-4 adjacency matrix

$$\begin{pmatrix} 0 & 1 & -1 & -1 \\ 1 & -1 & 1 & 1 \\ 0 & 1 & 1 & 2 \\ 0 & 1 & 3 & 0 \end{pmatrix} \tag{6.4}$$

where the two terminals correspond to the first and the fourth vertices. Its validity (for the value $b = 4$) can be verified by computing the permanent of the corresponding sub-matrices (see Exercise 6.25). ■

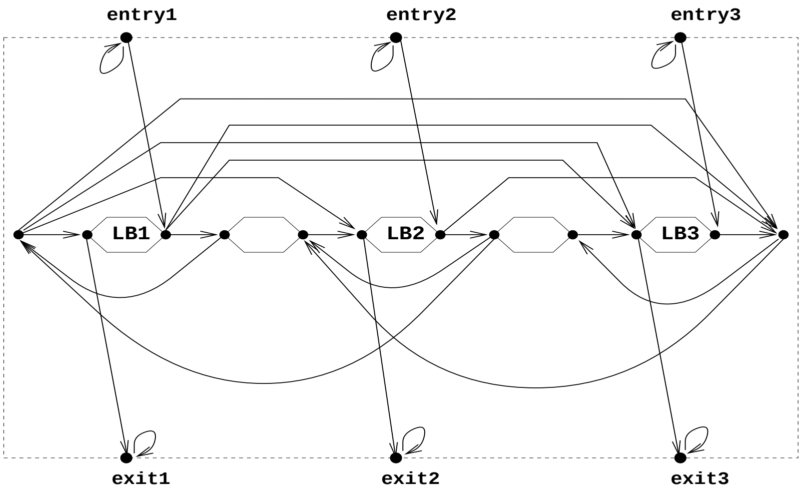
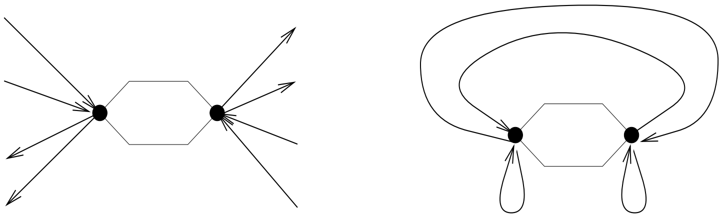


Figure 6.4: A structured clause gadget for the reduction to cycle cover.



On the left is a box with potential edges adjacent to it (as in the gadget construction). On the right is a box and the four external edges used in the analysis.

Figure 6.5: External edges for the analysis of the box

Proof of Proposition 6.20: The proof proceeds in two steps. In the first step we show that computing the permanent of *integer matrices* is reducible to computing the permanent of *non-negative matrices*. This reduction proceeds as follows. For an n-by-n integer matrix $A = (a_{i,j})_{i,j\in[n]}$, let $\|A\|_\infty = \max_{i,j}(|a_{i,j}|)$ and $Q_A = 2(n!)\cdot\|A\|_\infty^n + 1$. We note that, given A, the value Q_A can be computed in polynomial-time, and in particular $\log_2 Q_A < n^2 \log \|A\|_\infty$. Given the matrix A, the reduction constructs the non-negative matrix $A' = (a_{i,j} \bmod Q_A)_{i,j\in[n]}$ (i.e., the entries of A' are in $\{0,1,...,Q_A-1\}$), queries the oracle for the permanent of A', and outputs $v \stackrel{\rm def}{=} \texttt{perm}(A') \bmod Q_A$ if $v < Q_A/2$ and $-(Q_A - v)$ otherwise. The key observation is that

$$\texttt{perm}(A) \equiv \texttt{perm}(A') \pmod{Q_A}, \text{ while } |\texttt{perm}(A)| \leq (n!)\cdot\|A\|_\infty^n < Q_A/2.$$

Thus, $\texttt{perm}(A')$ mod Q_A (which is in $\{0, 1, ..., Q_A - 1\}$) determines $\texttt{perm}(A)$. We note that $\texttt{perm}(A')$ is likely to be much larger than $Q_A > |\texttt{perm}(A)|$; it is merely that $\texttt{perm}(A')$ and $\texttt{perm}(A)$ are equivalent modulo Q_A.

In the second step we show that computing the permanent of non-negative matrices is reducible to computing the permanent of 0/1-matrices. In this reduction, we view the computation of the permanent as the computation of the sum of the weights of the cycle covers (SWCC) of the corresponding weighted directed graph (see proof of Proposition 6.19). Thus, we reduce the computation of the SWCC of directed graphs *with non-negative weights* to the computation of the SWCC of *unweighted directed graphs with no parallel edges* (which correspond to 0/1-matrices). The reduction is via local replacements that preserve the value of the SWCC. These local replacements combine the following two local replacements (which preserve the SWCC):

1. Replacing an edge of weight $w = \prod_{i=1}^{t} w_i$ by a path of length t (i.e., $t-1$ internal nodes) with the corresponding weights $w_1, ..., w_t$, and self-loops (with weight 1) on all internal nodes.

   Note that a cycle-cover that uses the original edge corresponds to a cycle-cover that uses the entire path, whereas a cycle-cover that does not use the original edge corresponds to a cycle-cover that uses all the self-loops.

2. Replacing an edge of weight $w = \sum_{i=1}^{t} w_i$ by t parallel 2-edge paths such that the first edge on the i^{th} path has weight w_i, the second edge has weight 1, and the intermediate node has a self-loop (with weight 1). (Paths of length two are used because parallel edges are not allowed.)

   Note that a cycle-cover that uses the original edge corresponds to a collection of cycle-covers that use one out of the t paths (and the self-loops of all other intermediate nodes), whereas a cycle-cover that does not use the original edge corresponds to a cycle-cover that uses all the self-loops.

In particular, writing the positive integer w, having binary expansion $\sigma_{|w|-1} \cdots \sigma_0$, as $\sum_{i:\sigma_i=1}(1+1)^i$, we may apply the additive replacement (for the sum over $\{i : \sigma_i = 1\}$), next the product replacement (for each 2^i), and finally the additive replacement (for $1+1$). Applying this process to the matrix A' obtained in the first step, we *efficiently* obtain a matrix A'' with 0/1-entries such that $\texttt{perm}(A') = \texttt{perm}(A'')$. (In particular, the dimension of A'' is polynomial in the length of the binary representation of A', which in turn is polynomial in the length of the binary representation of A.) Combining the two reductions (steps), the proposition follows. ■

6.2.2 Approximate Counting

Having seen that exact counting (for relations in $\mathcal{PC}$) seems even harder than solving the corresponding search problems, we turn to relaxations of the counting problem. Before focusing on relative approximation, we briefly consider approximation with (large) additive deviation.

Let us consider the counting problem associated with an arbitrary $R \in \mathcal{PC}$. Without loss of generality, we assume that all solutions to n-bit instances have the same length $\ell(n)$, where indeed ℓ is a polynomial. We first note that, while it may be hard to compute $\#R$, given x it is easy to approximate $\#R(x)$ up to an additive error of $0.01 \cdot 2^{\ell(|x|)}$ (by randomly samplying potential solutions for x). Indeed, such an approximation is very rough, but it is not trivial (and in fact we do not know how to obatin it deterministically). In general, we can efficiently produce at random an estimate of $\#R(x)$ that, with high probability, deviates from the correct value by at most an additive term that is related to the absolute upperbound on the number of solutions (i.e., $2^{\ell(|x|)}$).

Proposition 6.21 (approximation with large additive deviation): *Let $R \in \mathcal{PC}$ and ℓ be a polynomial such that $R \subseteq \cup_{n\in\mathbb{N}}\{0,1\}^n \times \{0,1\}^{\ell(n)}$. Then, for every polynomial p, there exists a probabilistic polynomial-time algorithm A such that for every $x \in \{0,1\}^*$ and $\delta \in (0,1)$ it holds that*

$$\Pr[|A(x,\delta) - \#R(x)| > (1/p(|x|)) \cdot 2^{\ell(|x|)}] < \delta. \tag{6.5}$$

As usual, δ is presented to A in binary, and hence the running time of $A(x,\delta)$ is upper-bounded by $\mathrm{poly}(|x| \cdot \log(1/\delta))$.

Proof Sketch: On input x and δ, algorithm A sets $t = \Theta(p(|x|)^2 \cdot \log(1/\delta))$, selects uniformly $y_1, ..., y_t$ and outputs $|\{i : (x, y_i) \in R\}|/t$. □

Discussion. Proposition 6.21 is meaningful in the case that $\#R(x) > (1/p(|x|)) \cdot 2^{\ell(|x|)}$ holds for some x's. But otherwise, a trivial approximation (i.e., outputting the constant value zero) meets the bound of Eq. (6.5). In contrast to this notion of *additive approximation*, a *relative factor approximation* is typically more meaningful. Specifically, we will be interested in approximating $\#R(x)$ up-to a constant factor (or some other reasonable factor). In §6.2.2.1, we consider a natural $\#\mathcal{P}$-complete problem for which such a relative approximation can be obtained in probabilistic polynomial-time. We do not expect this to happen for every counting problem in $\#\mathcal{P}$, because a relative approximation allows for distinguishing instances having no solution from instances that do have solutions (i.e.,, deciding membership in S_R is reducible to a relative approximation of $\#R$). Thus, relative approximation for all $\#\mathcal{P}$ is at least as hard as deciding all problems in $\mathcal{NP}$, but in §6.2.2.2 we show that the former is not harder than the latter; that is, relative approximation for any problem in $\#\mathcal{P}$ can be obtained by a randomized Cook-reduction to $\mathcal{NP}$. Before turning to these results, let us state the underlying definition (and actually strengthen it by requiring approximation to within a factor of $1 \pm \varepsilon$, for $\varepsilon \in (0,1)$).[11]

[11] We refrain from formally defining an F-factor approximation, for an arbitrary F, although we shall refer to this notion in several informal discussions. There are several ways of defining the aforementioned term (and they are all equivalent when applied to our informal discussions). For example, an F-factor approximation of $\#R$ may mean that, with high probability, the output $A(x)$ satisfies $\#R(x)/F(|x|) \leq A(x) \leq F(|x|) \cdot \#R(x)$. Alternatively, we may require that $\#R(x) \leq A(x) \leq F(|x|) \cdot \#R(x)$ (or, alternatively, that $\#R(x)/F(|x|) \leq A(x) \leq \#R(x)$.

Definition 6.22 (approximation with relative deviation): *Let $f : \{0,1\}^* \to \mathbb{N}$ and $\varepsilon, \delta : \mathbb{N} \to [0,1]$. A randomized process Π is called an (ε, δ)-*approximator of f *if for every x it holds that*

$$\Pr[|\Pi(x) - f(x)| > \varepsilon(|x|) \cdot f(x)] \;<\; \delta(|x|). \tag{6.6}$$

We say that f is efficiently $(1-\varepsilon)$-approximable (or just $(1-\varepsilon)$-approximable) *if there exists a probabilistic polynomial-time algorithm A that constitute an $(\varepsilon, 1/3)$-approximator of f.*

The error probability of the latter algorithm A (which has error probability $1/3$) can be reduced to δ by $O(\log(1/\delta))$ repetitions (see Exercise 6.26). Typically, the running time of A will be polynomial in $1/\varepsilon$, and ε is called the deviation parameter.

6.2.2.1 Relative approximation for $\#R_{\mathtt{dnf}}$

In this subsection we present a natural $\#\mathcal{P}$-complete problem for which constant factor approximation can be found in probabilistic polynomial-time. Stronger results regarding unnatural problems appear in Exercise 6.27.

Consider the relation $R_{\mathtt{dnf}}$ consisting of pairs (ϕ, τ) such that ϕ is a DNF formula and τ is an assignment satisfying it. Recall that the search problem of $R_{\mathtt{dnf}}$ is easy to solve and that the proof of Theorem 6.17 establishes that $\#R_{\mathtt{dnf}}$ is $\#\mathcal{P}$-complete (via a non-parsimonious reduction). Still there exists a probabilistic polynomial-time algorithm that provides a constant factor approximation of $\#R_{\mathtt{dnf}}$. We warn that the fact that $\#R_{\mathtt{dnf}}$ is $\#\mathcal{P}$-complete *via a non-parsimonious reduction* means that the constant factor approximation for $\#R_{\mathtt{dnf}}$ does not seem to imply a similar approximation for all problems in $\#\mathcal{P}$. In fact, we should not expect each problem in $\#\mathcal{P}$ to have a (probabilistic) polynomial-time constant-factor approximation algorithm because this would imply $\mathcal{NP} \subseteq \mathcal{BPP}$ (since a constant factor approximation allows for distinguishing the case in which the instance has no solution from the case in which the instance has a solution).

The following algorithm is actually a deterministic reduction of the task of $(\varepsilon, 1/3)$-approximating $\#R_{\mathtt{dnf}}$ to an (additive deviation) approximation of the type provided in Proposition 6.21. Consider a DNF formula $\phi = \bigvee_{i=1}^m C_i$, where each $C_i : \{0,1\}^n \to \{0,1\}$ is a conjunction. Actually, we will deal with the more general problem in which we are (implicitly) given m subsets $S_1, ..., S_m \subseteq \{0,1\}^n$ and wish to approximate $|\bigcup_i S_i|$. In our case, each S_i is the set of assignments satisfying the conjunction C_i. In general, we make two computational assumptions regarding these sets (letting efficient mean implementable in time polynomial in $n \cdot m$):

1. Given $i \in [m]$, one can efficiently determine $|S_i|$.
2. Given $i \in [m]$ and $J \subseteq [m]$, one can efficiently approximate $\Pr_{s \in S_i}\left[s \in \bigcup_{j \in J} S_j\right]$ up to an *additive deviation of* $1/\mathrm{poly}(n+m)$.

These assumptions are satisfied in our setting (where $S_i = C_i^{-1}(1)$, see Exercise 6.28). Now, the key observation towards approximating $|\bigcup_{i=1}^m S_i|$ is that

$$\left|\bigcup_{i=1}^{m} S_i\right| = \sum_{i=1}^{m} \left|S_i \setminus \bigcup_{j<i} S_j\right| = \sum_{i=1}^{m} |S_i| \cdot \mathsf{Pr}_{s\in S_i}\left[s \notin \bigcup_{j<i} S_j\right] \tag{6.7}$$

and that the probabilities in Eq. (6.7) can be approximated by the second assumption. This leads to the following algorithm, where ε denotes the desired deviation parameter (i.e., we wish to obtain $(1 \pm \varepsilon) \cdot |\bigcup_{i=1}^m S_i|$).

Construction 6.23 *Let $\varepsilon' = \varepsilon/m$. For $i = 1$ to m do:*

1. *Using the first assumption, compute $|S_i|$.*
2. *Using the second assumption, obtain $\widetilde{p}_i = p_i \pm \varepsilon'$, where $p_i \stackrel{\text{def}}{=} \mathsf{Pr}_{s\in S_i}[s \notin \bigcup_{j<i} S_j]$. Set $a_i \stackrel{\text{def}}{=} \widetilde{p}_i \cdot |S_i|$.*

Output the sum of the a_i's.

Let $N_i = p_i \cdot |S_i|$. We are interested in the quality of the approximation to $\sum_i N_i = |\bigcup_i S_i|$ provided by $\sum_i a_i$. Using $a_i = (p_i \pm \varepsilon') \cdot |S_i| = N_i \pm \varepsilon' \cdot |S_i|$ (for all i's), we have $\sum_i a_i = \sum_i N_i \pm \varepsilon' \cdot \sum_i |S_i|$. Using $\sum_i |S_i| \leq m \cdot |\bigcup_i S_i| = m \cdot \sum_i N_i$ (and $\varepsilon = m\varepsilon'$), we get $\sum_i a_i = (1 \pm \varepsilon) \cdot \sum_i N_i$. Thus, we obtain the following result (see Exercise 6.28).

Proposition 6.24 *For every positive polynomial p, the counting problem of $R_{\mathtt{dnf}}$ is efficiently $(1 - (1/p))$-approximable.*

Using the reduction presented in the proof of Theorem 6.17, we conclude that the number of *un*satisfying assignments to a given CNF formula is efficiently $(1-(1/p))$-approximable. We warn, however, that the number of satisfying assignments to such a formula is *not* efficiently approximable. This concurs with the general phenomenon by which *relative approximation may be possible for one quantity, but not for the complementary quantity.* Needless to say, such a phenomenon does not occur in the context of additive-deviation approximation.

6.2.2.2 Relative approximation for $\#\mathcal{P}$

Recall that we cannot expect to efficiently approximate every $\#\mathcal{P}$ problem, where throughout the rest of this section "approximation" is used as a shorthand for "relative approximation" (as in Definition 6.22). Specifically, efficiently approximating $\#R$ yields an efficient algorithm for deciding membership in $S_R = \{x : R(x) \neq \emptyset\}$. Thus, at best we can hope that approximating $\#R$ is not harder than deciding S_R (i.e., that approximating $\#R$ is reducible in polynomial-time to S_R). This is indeed the case for every NP-complete problem (i.e., if S_R is NP-complete). More generally, we show that approximating any problem in $\#\mathcal{P}$ is reducible in probabilistic polynomial-time to $\mathcal{NP}$.

Theorem 6.25 *For every $R \in \mathcal{PC}$ and positive polynomial p, there exists a probabilistic polynomial-time oracle machine that when given oracle access to $\mathcal{NP}$ constitutes a $(1/p, \mu)$-approximator of $\#R$, where μ is a negligible function* (e.g., $\mu(n) = 2^{-n}$).

Recall that it suffices to provide a $(1/p, \delta)$-approximator of $\#R$, for any constant $\delta < 0.5$, because error reduction is applicable in this context (see Exercise 6.26). Also, it suffices to provide a $(1/2, \delta)$-approximator for every problem in $\#\mathcal{P}$ (see Exercise 6.29).

Proof: Given x, we show how to approximate $|R(x)|$ to within some constant factor. The desired $(1-(1/p))$-approximation can be obtained as in Exercise 6.29. We may also assume that $R(x) \neq \emptyset$, by starting with the query "is x in S_R" and halting (with output 0) if the answer is negative. Without loss of generality, we assume that $R(x) \subseteq \{0,1\}^\ell$, where $\ell = \mathrm{poly}(|x|)$. We focus on finding some $i \in \{1, ..., \ell\}$ such that $2^{i-4} \leq |R(x)| \leq 2^{i+4}$.

We proceed in iterations. For $i = 1, ..., \ell + 1$, we find out whether or not $|R(x)| < 2^i$. If the answer is positive then we halt with output 2^i, and otherwise we proceed to the next iteration. (Indeed, if we were able to obtain correct answers to these queries then the output 2^i would satisfy $2^{i-1} \leq |R(x)| < 2^i$.)

Needless to say, the key issue is how to check whether $|R(x)| < 2^i$. The main idea is to use a "random sieve" on the set $R(x)$ such that each element passes the sieve with probability 2^{-i}. Thus, we expect $|R(x)|/2^i$ elements of $R(x)$ to pass the sieve. Assuming that the number of elements in $R(x)$ that pass the random sieve is indeed $\lfloor |R(x)|/2^i \rfloor$, it holds that $|R(x)| \geq 2^i$ *if and only if some element of $R(x)$ passes the sieve.* Assuming that the sieve can be implemented efficiently, the question of whether or not some element in $R(x)$ passed the sieve is of an "NP-type" (and thus can be referred to our NP-oracle). Combining both assumptions, we may implement the foregoing process by proceeding to the next iteration as long as some element of $R(x)$ passes the sieve. Furthermore, this implementation will provide a reasonably good approximation even if the number of elements in $R(x)$ that pass the random sieve is only approximately equal to $|R(x)|/2^i$. In fact, the level of approximation that this implementation provides is closely related to the level of approximation that is provided by the random sieve. Details follow.

Implementing a random sieve. The random sieve is implemented by using a family of hashing functions (see Section D.2). Specifically, in the i^{th} iteration we use a family H_ℓ^i such that each $h \in H_\ell^i$ has a $\mathrm{poly}(\ell)$-bit long description and maps ℓ-bit long strings to i-bit long strings. Furthermore, the family is accompanied with an efficient evaluation algorithm (i.e., mapping adequate pairs (h, x) to $h(x)$) and satisfies (for every $S \subseteq \{0,1\}^\ell$)

$$\mathsf{Pr}_{h \in H_\ell^i}[|\{y \in S : h(y) = 0^i\}| \notin (1 \pm \varepsilon) \cdot 2^{-i}|S|] \ < \ \frac{2^i}{\varepsilon^2 |S|} \tag{6.8}$$

(see Lemma D.4). *The random sieve will let y pass if and only if $h(y) = 0^i$.* Indeed, this random sieve is not as perfect as we assumed in the foregoing discussion, but Eq. (6.8) suggests that in some sense this sieve is good enough.

Implementing the queries. Recall that for some x, i and $h \in H_\ell^i$, we need to determine whether $\{y \in R(x) : h(y) = 0^i\} = \emptyset$. This type of question can be cast as membership in the set

$$S_{R,H} \stackrel{\text{def}}{=} \{(x,i,h) : \exists y \text{ s.t. } (x,y) \in R \wedge h(y) = 0^i\}. \tag{6.9}$$

Using the hypotheses that $R \in \mathcal{PC}$ and that the family of hashing functions has an efficient evaluation algorithm, it follows that $S_{R,H}$ is in $\mathcal{NP}$.

The actual procedure. On input $x \in S_R$ and oracle access to $S_{R,H}$, we proceed in iterations, starting with $i = 1$ and halting at $i = \ell$ (if not before), where ℓ denotes the length of the potential solutions for x. In the i^{th} iteration (where $i < \ell$), we uniformly select $h \in H_\ell^i$ and query the oracle on whether or not $(x,i,h) \in S_{R,H}$. If the answer is negative then we halt with output 2^i, and otherwise we proceed to the next iteration (using $i \leftarrow i+1$). Needless to say, if we reach the last iteration (i.e., $i = \ell$) then we just halt with output 2^ℓ.

Indeed, we have ignored the case that $x \notin S_R$, which can be easily handled by a minor modification of the foregoing procedure. Specifically, on input x, we first query S_R on x and halt with output 0 if the answer is negative. Otherwise we proceed as in the foregoing procedure.

The analysis. We upper-bound separately the probability that the procedure outputs a value that is too small and the probability that it outputs a value that is too big. In light of the foregoing discussion, we may assume that $|R(x)| > 0$, and let $i_x = \lfloor \log_2 |R(x)| \rfloor \geq 0$.

1. The probability that the procedure *halts in a specific iteration* $i < i_x$ equals $\mathsf{Pr}_{h \in H_\ell^i}[|\{y \in R(x) : h(y) = 0^i\}| = 0]$, which in turn is upper-bounded by $2^i/|R(x)|$ (using Eq. (6.8) with $\varepsilon = 1$). Thus, the probability that the procedure halts *before* iteration $i_x - 3$ is upper-bounded by $\sum_{i=0}^{i_x-4} 2^i/|R(x)|$, which in turn is less than $1/8$ (because $i_x \leq \log_2 |R(x)|$). Thus, with probability at least $7/8$, the output is at least $2^{i_x-3} > |R(x)|/16$ (because $i_x > (\log_2 |R(x)|) - 1$).

2. The probability that the procedure *does not halt in iteration* $i > i_x$ equals $\mathsf{Pr}_{h \in H_\ell^i}[|\{y \in R(x) : h(y) = 0^i\}| \geq 1]$, which in turn is upper-bounded by $\alpha/(\alpha-1)^2$, where $\alpha = 2^i/|R(x)| > 1$ (using Eq. (6.8) with $\varepsilon = \alpha - 1$).[12] Thus, the probability that the procedure does not halt by iteration $i_x + 4$ is upper-bounded by $8/49 < 1/6$ (because $i_x > (\log_2 |R(x)|) - 1$). Thus, with probability at least $5/6$, the output is at most $2^{i_x+4} \leq 16 \cdot |R(x)|$ (because $i_x \leq \log_2 |R(x)|$).

Thus, with probability at least $(7/8)-(1/6) > 2/3$, the foregoing procedure outputs a value v such that $v/16 \leq |R(x)| < 16v$. Reducing the deviation by using the ideas presented in Exercise 6.29 (and reducing the error probability as in Exercise 6.26), the theorem follows. ■

[12] A better bound can be obtained by using the hypothesis that, for every y, when h is uniformly selected in H_ℓ^i, the value of $h(y)$ is uniformly distributed in $\{0,1\}^i$. In this case, $\mathsf{Pr}_{h \in H_\ell^i}[|\{y \in R(x) : h(y) = 0^i\}| \geq 1]$ is upper-bounded by $\mathsf{E}_{h \in H_\ell^i}[|\{y \in R(x) : h(y) = 0^i\}|] = |R(x)|/2^i$.

Perspective. The key observation underlying the proof Theorem 6.25 is that, while (even with the help of an NP-oracle) we cannot directly test whether the number of solutions is greater than a given number, we can test (with the help of an NP-oracle) whether the number of solutions that "survive a random sieve" is greater than zero. If fact, we can also test whether the number of solutions that "survive a random sieve" is greater than a small number, where small means polynomial in the length of the input (see Exercise 6.31). That is, the complexity of this test is linear in the size of the threshold, and not in the length of its binary description. Indeed, in many settings it is more advantageous to use a threshold that is polynomial in some efficiency parameter (rather than using the threshold zero); examples appear in §6.2.4.2 and in [102].

6.2.3 Searching for unique solutions

A natural computational problem (regarding search problems), which arises when discussing the number of solutions, is the problem of distinguishing instances having a single solution from instances having no solution (or finding the unique solution whenever such exists). We mention that instances having a single solution facilitate numerous arguments (see, for example, Exercise 6.21 and §10.2.2.1). Formally, searching for and deciding the existence of unique solutions are defined within the framework of promise problems (see Section 2.4.1).

Definition 6.26 (search and decision problems for unique solution instances): *The* set of instances having unique solutions *with respect to the binary relation R is defined as* $\mathtt{US}_R \stackrel{\rm def}{=} \{x : |R(x)| = 1\}$, *where* $R(x) \stackrel{\rm def}{=} \{y : (x,y)\in R\}$. *As usual, we denote* $S_R = \{x : |R(x)| \geq 1\}$, *and* $\overline{S}_R \stackrel{\rm def}{=} \{0,1\}^* \setminus S_R = \{x : |R(x)| = 0\}$.

- *The problem of* finding unique solutions for R *is defined as the* search problem R with promise $\mathtt{US}_R \cup \overline{S}_R$ (see Definition 2.28).

  In continuation to Definition 2.29, the candid searching for unique solutions for R *is defined as the* search problem R with promise $\mathtt{US}_R$.

- *The problem of* deciding unique solution for R *is defined as the* promise problem $(\mathtt{US}_R, \overline{S}_R)$ (see Definition 2.30).

Interestingly, in many natural cases, the promise does not make any of these problems any easier than the original problem. That is, for all known NP-complete problems, the original problem is reducible in probabilistic polynomial-time to the corresponding unique instances problem.

Theorem 6.27 *Let $R \in \mathcal{PC}$ and suppose that every search problem in $\mathcal{PC}$ is parsimoniously reducible to R. Then solving the search problem of R* (resp., deciding membership in S_R) *is reducible in probabilistic polynomial-time to finding unique solutions for R* (resp., to the promise problem $(\mathtt{US}_R, \overline{S}_R)$). *Furthermore, there exists a probabilistic polynomial-time computable mapping M such that for every $x \in \overline{S}_R$ it holds that $M(x) \in \overline{S}_R$, whereas for every $x \in S_R$ it holds that* $\mathsf{Pr}[M(x) \in \mathtt{US}_R] \geq 1/\mathrm{poly}(|x|)$.

We highlight the hypothesis that R is $\mathcal{PC}$-complete *via parsimonious reductions* is crucial to Theorem 6.27 (see Exercise 6.32). The large (but bounded-away from 1) error probability of the randomized Karp-reduction M can be reduced by repetitions, yielding a randomized Cook-reduction with exponentially vanishing error probability. Note that the resulting reduction may make many queries that violate the promise, and still yields the correct answer (with high probability) by relying on queries that satisfy the promise. (Specifically, in the case of search problems we avoid wrong solutions by checking each solution obtained, while in the case of decision problems we rely on the fact that for every $x \in \overline{S}_R$ it holds that $M(x) \in \overline{S}_R$.)

Proof: As in the proof of Theorem 6.25, the idea is to apply a "random sieve" on $R(x)$, this time with the hope that a single element survives. Specifically, if we let each element passes the sieve with probability approximately $1/|R(x)|$ then with constant probability a single element survives (and we shall obtain an instance with a unique solution). Sieving will be performed by a random function selected in an adequate hashing family (see Section D.2). A couple of questions arise:

1. *How do we get an approximation to* $|R(x)|$? Note that we need such an approximation in order to determine the adequate hashing family. Indeed, we may just invoke Theorem 6.25, but this will not yield a many-to-one reduction. Instead, we just select $m \in \{0, ..., \text{poly}(|x|)\}$ uniformly and note that (if $|R(x)| > 0$ then) $\Pr[m = \lceil \log_2 |R(x)| \rceil] = 1/\text{poly}(|x|)$. Next, we randomly map x to (x, m, h), where h is uniformly selected in an adequate hashing family.

2. *How does the question of whether a single element of* $R(x)$ *pass the random sieve translate to an instance of the unique-solution problem for* R? Recall that in the proof of Theorem 6.25 the non-emptiness of the set of element of $R(x)$ that pass the sieve (defined by h) was determined by checking membership (of (x, m, h)) in $S_{R,H} \in \mathcal{NP}$ (defined in Eq. (6.9)). Furthermore, the number of NP-witnesses for $(x, m, h) \in S_{R,H}$ equals the number of elements of $R(x)$ that pass the sieve. Using the parsimonious reduction of $S_{R,H}$ to S_R (which is guaranteed by the theorem's hypothesis), we obtained the desired instance.

Note that in case $R(x) = \emptyset$ the aforementioned mapping always generates a no-instance (of $S_{R,H}$ and thus of S_R). Details follow.

Implementation (i.e., the mapping M). As in the proof of Theorem 6.25, we assume, without loss of generality, that $R(x) \subseteq \{0,1\}^\ell$, where $\ell = \text{poly}(|x|)$. We start by uniformly selecting $m \in \{1, ..., \ell + 1\}$ and $h \in H_\ell^m$, where H_ℓ^m is a family of efficiently computable and pairwise-independent hashing functions (see Definition D.1) mapping ℓ-bit long strings to m-bit long strings. Thus, we obtain an instance (x, m, h) of $S_{R,H} \in \mathcal{NP}$ such that the set of valid solutions for (x, m, h) equals $\{y \in R(x) : h(y) = 0^m\}$. Using the parsimonious reduction g of $S_{R,H}$ to S_R, we map (x, m, h) to $g(x, m, h)$, and it holds that $|\{y \in R(x) : h(y) = 0^m\}|$ equals $|R(g(x, m, h))|$. To summarize, on input x the randomized mapping M outputs the instance $M(x) \stackrel{\rm def}{=} g(x,m,h)$, where $m \in \{1,...,\ell+1\}$ and $h \in H_\ell^m$ are uniformly selected.

The analysis. Note that for any $x \in \overline{S}_R$ it holds that $\Pr[M(x) \in \overline{S}_R] = 1$. Assuming that $x \in S_R$, with probability exactly $1/(\ell+1)$ it holds that $m = m_x$, where $m_x \stackrel{\rm def}{=} \lceil \log_2 |R(x)| \rceil + 1$. In this case, for a uniformly selected $h \in H_\ell^m$, we lower-bound the probability that $\{y \in R(x) : h(y) = 0^m\}$ is a singleton. Using the Inclusion-Exclusion Principle, we have

$$\begin{aligned}
&\Pr_{h\in H_\ell^{m_x}}[|\{y\in R(x) : h(y)=0^{m_x}\}| = 1] \qquad (6.10)\\
&= \Pr_{h\in H_\ell^{m_x}}[|\{y\in R(x) : h(y)=0^{m_x}\}| > 0] \;-\; \Pr_{h\in H_\ell^{m_x}}[|\{y\in R(x) : h(y)=0^{m_x}\}| > 1]\\
&\geq \sum_{y\in R(x)} \Pr_{h\in H_\ell^{m_x}}[h(y)=0^{m_x}] \;-\; 2\cdot \sum_{y_1<y_2\in R(x)} \Pr_{h\in H_\ell^{m_x}}[h(y_1)=h(y_2)=0^{m_x}]\\
&= |R(x)|\cdot 2^{-m_x} - 2\cdot \binom{|R(x)|}{2}\cdot 2^{-2m_x}
\end{aligned}$$

where the last equality is due to the pairwise independence property. Using $2^{m_x-2} < |R(x)| \leq 2^{m_x-1}$, it follows that Eq. (6.10) is lower-bounded by 1/4. Thus, $\Pr[M(x) \in \mathtt{US}_R] \geq 1/4(\ell+1)$, and the theorem follows. ∎

Comment. Theorem 6.27 is sometimes stated as referring to the unique solution problem of SAT. In this case and when using a specific family of pairwise independent hashing functions, the use of the parsimonious reduction can be avoided. For details see Exercise 6.33.

6.2.4 Uniform generation of solutions

We now turn to a new type of computational problems, which may be viewed as a straining of search problems. We refer to the task of generating a uniformly distributed solution for a given instance, rather than merely finding an adequate solution. Needless to say, by definition, algorithms solving this ("uniform generation") task must be randomized. Focusing on relations in $\mathcal{PC}$ we consider two versions of the problem, which differ by the level of approximation provided for the desired (uniform) distribution.[13]

Definition 6.28 (uniform generation): *Let $R \in \mathcal{PC}$ and $S_R = \{x : |R(x)| \geq 1\}$, and let Π be a probabilistic process.*

1. *We say that Π* solves the uniform generation problem of *R if, on input $x \in S_R$, the process Π outputs either an element of $R(x)$ or a special symbol, denoted $\perp$, such that $\Pr[\Pi(x) \in R(x)] \geq 1/2$ and for every $y \in R(x)$ it holds that $\Pr[\Pi(x) = y \mid \Pi(x) \in R(x)] = 1/|R(x)|$.*

[13]Note that a probabilistic algorithm running in strict polynomial-time is not able to output a perfectly uniform distribution on sets of certain sizes. Specifically, referring to the standard model that allows only for uniformly selected binary values, such algorithms cannot output a perfectly uniform distribution on sets having cardinality that is not a power of two.

2. *For $\varepsilon : \mathbb{N} \to [0,1]$, we say that Π* solves the $(1-\varepsilon)$-approximate uniform generation problem of R *if, on input $x \in S_R$, the distribution $\Pi(x)$ is $\varepsilon(|x|)$-close*[14] *to the uniform distribution on $R(x)$.*

In both cases, without loss of generality, we may require that if $x \notin S_R$ then $\Pr[\Pi(x) = \perp] = 1$. *More generally, we may require that Π never outputs a string not in $R(x)$.*

Note that the error probability of uniform generation (as in Item 1) can be made exponentially vanishing (in $|x|$) by employing error-reduction. In contrast, we are not aware of any general way of reducing the deviation of an approximate uniform generation procedure (as in Item 2).[15]

In §6.2.4.1 we show that, for many search problems, approximate uniform generation is computationally equivalent to approximate counting. In §6.2.4.2 we present a direct approach for solving the uniform generation problem of any search problem in $\mathcal{PC}$ by using an oracle to $\mathcal{NP}$.

6.2.4.1 Relation to approximate counting

We show that, for many natural search problems in $\mathcal{PC}$, the approximate counting problem associated with R is computationally equivalent to approximate uniform generation with respect to R. Specifically, we refer to search problems $R \in \mathcal{PC}$ such that $R'(x;y') \stackrel{\rm def}{=} \{y'' : (x,y'y'') \in R\}$ is *strongly parsimoniously reducible to* R, where a strongly parsimonious reduction of R' to R is a parsimonious reduction g that is coupled with an efficiently computable 1-1 mapping of pairs $(g(x),y) \in R$ to pairs $(x,h(x,y)) \in R'$ (i.e., h is efficiently computable and $h(x,\cdot)$ is a 1-1 mapping of $R(g(x))$ to $R'(x)$). Note that for many natural search problems R (e.g., the search problem of `SAT` and `Perfect Matching`), the corresponding R' is strongly parsimoniously reducible to R.

Recalling that both types of approximation problems are parameterized by the level of precision, we obtain the following quantitative form of the aforementioned equivalence.

Theorem 6.29 *Let $R \in \mathcal{PC}$ and let ℓ be a polynomial such that for every $(x,y) \in R$ it holds that $|y| \leq \ell(|x|)$. Suppose that R' is strongly parsimoniously reducible to R, where $R'(x;y') \stackrel{\rm def}{=} \{y'' : (x,y'y'') \in R\}$.*

1. From approximate counting to approximate uniform generation: *Let $\varepsilon(n) = 1/5\ell(n)$ and let $\mu : \mathbb{N} \to (0,1)$ be a function satisfying $\mu(n) \geq \exp(-\mathrm{poly}(n))$. Then, $(1-\mu)$-approximate uniform generation for R is reducible in probabilistic polynomial-time to $(1-\varepsilon)$-approximating $\#R$.*

2. From approximate uniform generation to approximate counting: *For every noticeable $\varepsilon : \mathbb{N} \to (0,1)$* (i.e., $\varepsilon(n) \geq 1/\mathrm{poly}(n)$ for every n), *the problem of*

[14] See Section D.1.1.

[15] We note that in some cases, the deviation of an approximate uniform generation procedure can be reduced. See discussion following Theorem 6.29.

> $(1-\varepsilon)$*-approximating* $\#R$ *is reducible in probabilistic polynomial-time to* $(1-\varepsilon')$*-approximate uniform generation problem of* R*, where* $\varepsilon'(n)=\varepsilon(n)/5\ell(n)$.

In fact, Part 1 holds also in case R' *is just parsimoniously reducible to* R.

Note that the quality of the approximate uniform generation asserted in Part 1 (i.e., μ) is independent of the quality of the approximate counting procedure (i.e., ε) to which the former is reduced, provided that the approximate counter performs better than some threshold. On the other hand, the quality of the approximate counting asserted in Part 2 (i.e., ε) does depend on the quality of the approximate uniform generation (i.e., ε'), but cannot reach beyond a certain bound (i.e., noticeable relative devaition). Recall, that for problems that are NP-complete under parsimonious reductions the quality of approximate counting procedures can be improved (see Exercise 6.30). However, Theorem 6.29 is most useful when applied to problems that are not NP-complete, because for problems that are NP-complete both approximate counting and uniform generation are randomly reducible to the corresponding search problem (see Exercise 6.35).

Proof: Throughout the proof, we assume for simplicity (and in fact without loss of generality) that $R(x)\neq\emptyset$ and $R(x)\subseteq\{0,1\}^{\ell(|x|)}$.

Towards Part 1, let us first reduce the uniform generation problem of R to $\#R$ (rather than to approximating $\#R$). On input $x\in S_R$, we generate a uniformly distributed $y\in R(x)$ by randomly generating its bits one after the other. We proceed in iterations, entering the i^{th} iteration with an $(i-1)$-bit long string y' such that $R'(x;y')\stackrel{\text{def}}{=}\{y'':(x,y'y'')\in R\}$ is not empty. With probability $|R'(x;y'1)|/|R'(x;y')|$ we set the i^{th} bit to equal 1, and otherwise we set it to equal 0. We obtain both $|R'(x;y'1)|$ and $|R'(x;y')|$ by using a parsimonious reduction g of $R'=\{((x;y'),y''):(x,y'y'')\in R\}\in\mathcal{PC}$ to R. That is, we obtain $|R'(x;y')|$ by querying for the value of $|R(g(x;y'))|$. Ignoring integrality issues, all this works perfectly (i.e., we generate an $\ell(n)$-bit string uniformly distributed in $R(x)$) as long as we have oracle access to $\#R$. But we only have oracle access to an approximation of $\#R$, and thus a careful modification is in place.

Let us denote the approximation oracle by A. Firstly, by adequate error reduction, we may assume that, for every x, it holds that $\Pr[A(x)\in(1\pm\varepsilon(n))\cdot\#R(x)]>1-\mu'(|x|)$, where $\mu'(n)=\mu(n)/\ell(n)$. In the rest of the analysis we ignore the probability that the estimate deviates from the aforementioned interval, and note that this rare event is the only source of the possible deviation of the output distribution from the uniform distribution on $R(x)$.[16] Let us assume for a moment that A is *deterministic* and that for every x and y' it holds that

$$A(g(x,y'0))+A(g(x,y'1))\ \leq\ A(g(x;y')). \tag{6.11}$$

We also assume that the approximation is correct at the "trivial level" (where one may just check whether or not (x,y) is in R); that is, for every $y\in\{0,1\}^{\ell(|x|)}$, it

[16] The possible deviation is due to the fact that this rare event may occur with different probability in the different invocations of algorithm A.

holds that

$$A(g(x;y)) = 1 \text{ if } (x,y) \in R \text{ and } A(g(x;y)) = 0 \text{ otherwise.} \tag{6.12}$$

We modify the i^{th} iteration of the foregoing procedure such that, when entering with the $(i-1)$-bit long prefix y', we set the i^{th} bit to $\sigma \in \{0,1\}$ with probability $A(g(x;y'\sigma))/A(g(x;y'))$ and halt (with output $\perp$) with the residual probability (i.e., $1-(A(g(x;y'0))/A(g(x;y'))) - (A(g(x;y'1))/A(g(x;y')))$). Indeed, Eq. (6.11) guarantees that the latter instruction is sound, since the two main probabilities sum-up to at most 1. If we completed the last (i.e., $\ell(|x|)^{\text{th}}$) iteration, then we output the $\ell(|x|)$-bit long string that was generated. Thus, as long as Eq. (6.11) holds (but regardless of other aspects of the quality of the approximation), every $y = \sigma_1 \cdots \sigma_{\ell(|x|)} \in R(x)$, is output with probability

$$\frac{A(g(x;\sigma_1))}{A(g(x;\lambda))} \cdot \frac{A(g(x;\sigma_1\sigma_2))}{A(g(x;\sigma_1))} \cdots \frac{A(g(x;\sigma_1\sigma_2\cdots\sigma_{\ell(|x|)}))}{A(g(x;\sigma_1\sigma_2\cdots\sigma_{\ell(|x|)-1}))} \tag{6.13}$$

which, by Eq. (6.12), equals $1/A(g(x;\lambda))$. Thus, the procedure outputs each element of $R(x)$ with equal probability, and never outputs a non-$\perp$ value that is outside $R(x)$. It follows that the quality of approximation only effects the probability that the procedure outputs a non-$\perp$ value (which in turn equals $|R(x)|/A(g(x;\lambda))$). The key point is that, as long as Eq. (6.12) holds, the specific approximate values obtained by the procedure are immaterial – with the exception of $A(g(x;\lambda))$, all these values "cancel out".

We now turn to enforcing Eq. (6.11) and Eq. (6.12). We may enforce Eq. (6.12) by performing the straightforward check (of whether or not $(x,y) \in R$) rather than invoking $A(g(x,y))$.[17] As for Eq. (6.11), we enforce it artificially by using $A'(x,y') \stackrel{\text{def}}{=} (1+\varepsilon(|x|))^{3(\ell(|x|)-|y'|)} \cdot A(g(x;y'))$ instead of $A(g(x;y'))$. Recalling that $A(g(x;y')) = (1 \pm \varepsilon(|xy'|)) \cdot |R'(x;y')|$, we have

$$\begin{aligned} A'(x,y') &> (1+\varepsilon(|x|))^{3(\ell(|x|)-|y'|)} \cdot (1-\varepsilon(|x|)) \cdot |R'(x;y')| \\ A'(x,y'\sigma) &< (1+\varepsilon(|x|))^{3(\ell(|x|)-|y'|-1)} \cdot (1+\varepsilon(|x|)) \cdot |R'(x;y'\sigma)| \end{aligned}$$

and the claim follows using $(1-\varepsilon(|x|)) \cdot (1+\varepsilon(|x|))^3 > (1-\varepsilon(|x|))$. Note that the foregoing modification only decreases the probability of outputting a non-$\perp$ value by a factor of $(1+\varepsilon(|x|))^{3\ell(|x|)} < 2$, where the inequality is due to the setting of ε (i.e., $\varepsilon(n) = 1/5\ell(n)$). Finally, we refer to our assumption that A is deterministic. This assumption was only used in order to identify the value of $A(g(x,y'))$ obtained and used in the $(|y'|-1)^{\text{st}}$ iteration with the value of $A(g(x,y'))$ obtained and used in the $|y'|^{\text{th}}$ iteration, but the same effect can be obtained by just re-using the former value (in the $|y'|^{\text{th}}$ iteration) rather than re-invoking A in order to obtain it. Part 1 follows.

Towards Part 2, let use first reduce the task of approximating $\#R$ to the task of (exact) uniform generation for R. On input $x \in S_R$, the reduction uses

[17]Alternatively, we note that since A is a $(1-\varepsilon)$-approximator for $\varepsilon < 1$ it must hold that $\#R'(z) = 0$ implies $A(z) = 0$. Also, since $\varepsilon < 1/3$, if $\#R'(z) = 1$ then $A(z) \in (2/3, 4/3)$, which may be rounded to 1.

the tree of possible prefixes of elements of $R(x)$ in a somewhat different manner. Again, we proceed in iterations, entering the i^{th} iteration with an $(i-1)$-bit long string y' such that $R'(x;y') \stackrel{\text{def}}{=} \{y'' : (x, y'y'') \in R\}$ is not empty. At the i^{th} iteration we estimate the bigger among the two fractions $|R'(x;y'0)|/|R'(x;y')|$ and $|R'(x;y'1)|/|R'(x;y')|$, by uniformly sampling the uniform distribution over $R'(x;y')$. That is, taking $\text{poly}(|x|/\varepsilon'(|x|))$ uniformly distributed samples in $R'(x;y')$, we obtain with overwhelmingly high probability an approximation of these fractions up to an additive deviation of at most $\varepsilon'(|x|)/3$. This means that we obtain a relative approximation up-to a factor of $1 \pm \varepsilon'(|x|)$ for the fraction (or fractions) that is (resp., are) bigger than $1/3$. Indeed, we may not be able to obtain such a good relative approximation of the other fraction (in case it is very small), but this does not matter. It also does not matter that we cannot tell which is the bigger fraction among the two; it only matter that we use an approximation that indicates a quantity that is, say, bigger than $1/3$. We proceed to the next iteration by augmenting y' using the bit that corresponds to such a quantity. Specifically, suppose that we obtained the approximations $a_0(y') \approx |R'(x;y'0)|/|R'(x;y')|$ and $a_1(y') \approx |R'(x;y'1)|/|R'(x;y')|$. Then we extend y' by the bit 1 if $a_1(y') > a_0(y')$ and extend y' by the bit 0 otherwise. Finally, when we reach $y = \sigma_1 \cdots \sigma_{\ell(|x|)}$ such that $(x, y) \in R$, we output

$$a_{\sigma_1}(\lambda)^{-1} \cdot a_{\sigma_2}(\sigma_1)^{-1} \cdots a_{\sigma_{\ell(|x|)}}(\sigma_1\sigma_2 \cdots \sigma_{\ell(|x|)-1})^{-1}. \tag{6.14}$$

As in Part 1, actions regarding R' (in this case uniform generation in R') are conducted via the parsimonious reduction g to R. That is, whenever we need to sample uniformly in the set $R'(x;y')$, we sample the set $R(g(x;y'))$ and recover the corresponding element of $R'(x;y')$ by using the mapping guaranteed by the hypothesis that g is strongly parsimonious. Finally, note that the deviation from uniform distribution (i.e., the fact that we can only approximately sample R) merely introduces such a deviation in each of our approximations to the relevant fractions (i.e., to a fraction bigger than $1/3$). Specifically, on input x, using an oracle that provides a $(1-\varepsilon')$-approximate uniform generation for R, with overwhelmingly high probability, the output (as defined in Eq. (6.14)) is in

$$\prod_{i=1}^{\ell(|x|)} \left((1 \pm 2\varepsilon'(|x|)) \cdot \frac{|R'(x;\sigma_1 \cdots \sigma_{i-1})|}{|R'(x;\sigma_1 \cdots \sigma_i)|} \right) \tag{6.15}$$

where the error probability is due to the unlikely case that in one of the iterations our approximations deviates from the correct value by more than an additive deviation term of $\varepsilon'(n)/3$. Noting that Eq. (6.15) equals $(1 \pm 2\varepsilon'(|x|))^{\ell(|x|)} \cdot |R(x)|$ and using $(1 \pm 2\varepsilon'(|x|))^{\ell(|x|)} \subset (1 \pm \varepsilon(|x|))$, Part 2 follows, and so does the theorem. ■

6.2.4.2 A direct procedure for uniform generation

We conclude the current chapter by presenting a direct procedure for solving the uniform generation problem of any $R \in \mathcal{PC}$. This procedure uses an oracle to

$\mathcal{NP}$, which is unavoidable because solving the uniform generation problem implies solving the corresponding search problem. One advantage of this procedure, over the reduction presented in §6.2.4.1, is that it solves the uniform generation problem rather than the *approximate* uniform generation problem.

We are going to use hashing again, but this time we use a family of hashing functions having a stronger "uniformity property" (see Section D.2.3). Specifically, we will use a family of ℓ-wise independent hashing functions mapping ℓ-bit strings to m-bit strings, where ℓ bounds the length of solutions in R, and rely on the fact that such a family satisfies Lemma D.6. Intuitively, such functions partition $\{0,1\}^\ell$ into 2^m cells and Lemma D.6 asserts that these partitions "uniformly shatter" all sufficiently large sets. That is, for every set $S \subseteq \{0,1\}^\ell$ of size $\Omega(\ell \cdot 2^m)$ the partition induced by almost every function is such that each cell contains approximately $|S|/2^m$ elements of S. In particular, if $|S| = \Theta(\ell \cdot 2^m)$ then each cell contains $\Theta(\ell)$ elements of S.

Loosely speaking, the following procedure (for uniform generation) first selects a random hashing function and tests whether it "uniformly shatters" the target set S. If this condition holds then the procedure selects a cell at random and retrieve the elements of S residing in the chosen cell. Finally, the procedure outputs each retrieves element (in S) with a fixed probability, which is independent of the actual number of elements of S that reside in the chosen cell. This guarantees that each element $e \in S$ is output with the same probability, regardless of the number of elements of S that resides with e in the same cell.

In the following construction, we assume that on input x we also obtain a good approximation to the size of $R(x)$. This assumption can be enforced by using an approximate counting procedure as a preprocessing stage. Alternatively, the ideas presented in the following construction yield such an approximate counting procedure.

Construction 6.30 (uniform generation): *On input x and $m'_x \in \{m_x, m_x + 1\}$, where $m_x \stackrel{\text{def}}{=} \lfloor \log_2 |R(x)| \rfloor$ and $R(x) \subseteq \{0,1\}^\ell$, the oracle machine proceeds as follows.*

1. Selecting a partition that "uniformly shatters" $R(x)$. *The machine sets $m = \max(0, m'_x - 6 - \log_2 \ell)$ and selects uniformly $h \in H_\ell^m$. Such a function defines a partition of $\{0,1\}^\ell$ into 2^m cells*[18]*, and the hope is that each cell contains approximately the same number of elements of $R(x)$. Next, the machine checks that this is indeed the case or rather than no cell contains more that 128ℓ elements of $R(x)$. This is done by checking whether or not $(x, h, 1^{128\ell+1})$ is in the set $S_{R,H}^{(1)}$ defined as follows*

$$\begin{aligned} S_{R,H}^{(1)} &\stackrel{\text{def}}{=} \{(x', h', 1^t) : \exists v \text{ s.t. } |\{y : (x', y) \in R \wedge h'(y) = v\}| \geq t\} \quad (6.16) \\ &= \{(x', h', 1^t) : \exists v, y_1, ..., y_t \text{ s.t. } \psi^{(1)}(x', h', v, y_1, ..., y_t)\}, \end{aligned}$$

[18]For sake of uniformity, we allow also the case of $m = 0$, which is rather artificial. In this case all hashing functions in H_ℓ^0 map $\{0,1\}^\ell$ to the empty string, which is viewed as 0^0, and thus define a trivial partition of $\{0,1\}^\ell$ (i.e., into a single cell).

where $\psi^{(1)}(x', h', v, y_1, ..., y_t)$ *holds if and only if* $y_1 < y_2 \cdots < y_t$ *and for every* $j \in [t]$ *it holds that* $(x', y_j) \in R \wedge h'(y_j) = v$. *Note that* $S^{(1)}_{R,H} \in \mathcal{NP}$.

If the answer is positive (i.e., there exists a cell that contains more that 128ℓ elements of $R(x)$) *then the machine halts with output* $\perp$. *Otherwise, the machine continues with this choice of* h. *In this case, no cell contains more that* 128ℓ *elements of* $R(x)$ (i.e., for every $v \in \{0,1\}^m$, it holds that $|\{y : (x,y) \in R \wedge h(y) = v\}| \leq 128\ell$). *We stress that this is an absolute guarantee that follows from* $(x, h, 1^{128\ell+1}) \notin S^{(1)}_{R,H}$.

2. Selecting a cell and determining the number of elements of $R(x)$ that are contained in it. *The machine selects uniformly* $v \in \{0,1\}^m$ *and determines* $s_v \stackrel{\text{def}}{=} |\{y : (x,y) \in R \wedge h(y) = v\}|$ *by making queries to the following NP-set*

$$S^{(2)}_{R,H} \stackrel{\text{def}}{=} \{(x', h', v', 1^t) : \exists y_1, ..., y_t \text{ s.t. } \psi^{(1)}(x', h', v', y_1, ..., y_t)\}. \qquad (6.17)$$

Specifically, for $i = 1, ..., 128\ell$, *it checks whether* $(x, h, v, 1^i)$ *is in* $S^{(2)}_{R,H}$, *and sets* s_v *to be the largest value of* i *for which the answer is positive.*

3. Obtaining all the elements of $R(x)$ that are contained in the selected cell, and outputting one of them at random. *Using* s_v, *the procedure reconstructs the set* $S_v \stackrel{\text{def}}{=} \{y : (x,y) \in R \wedge h(y) = v\}$, *by making queries to the following NP-set*

$$S^{(3)}_{R,H} \stackrel{\text{def}}{=} \{(x', h', v', 1^t, j) : \exists y_1, ..., y_t \text{ s.t. } \psi^{(3)}(x', h', v', y_1, ..., y_t, j)\}, \quad (6.18)$$

where $\psi^{(3)}(x', h', v', y_1, ..., y_t, j)$ *holds if and only if* $\psi^{(1)}(x', h', v', y_1, ..., y_t)$ *holds and the* j^{th} *bit of* $y_1 \cdots y_t$ *equals* 1. *Specifically, for* $j_1 = 1, ..., s_v$ *and* $j_2 = 1, ..., \ell$, *we make the query* $(x, h, v, 1^{s_v}, (j_1 - 1) \cdot \ell + j_2)$ *in order to determine the* j_2^{th} *bit of* y_{j_1}. *Finally, having recovered* S_v, *the procedure outputs each* $y \in S_v$ *with probability* $1/128\ell$, *and outputs* $\perp$ *otherwise (i.e., with probability* $1 - (s_v/128\ell)$).

Focusing on the case that $m'_x > 6 + \log_2 \ell$, we note that $m \leq m'_x - 6 - \log_2 \ell < \log_2(|R(x)|/20\ell)$. In this case, by Lemma D.6, with overwhelmingly high probability, each set $\{y : (x,y) \in R \wedge h(y) = v\}$ has cardinality $(1 \pm 0.5)|R(x)|/2^m$. Using $m'_x > (\log_2 |R(x)|) - 1$ (resp., $m'_x \leq (\log_2 |R(x)|) + 1$), it follows that $|R(x)|/2^m < 128\ell$ (resp., $|R(x)|/2^m \geq 16\ell$). Thus, Step 1 can be easily adapted to yield an approximate counting procedure for $\#R$ (see Exercise 6.34). However, our aim is to establish the following fact.

Proposition 6.31 *Construction 6.30 solves the uniform generation problem of* R.

Proof: By Lemma D.6 (and the setting of m), with overwhelmingly high probability, a uniformly selected $h \in H^m_\ell$ partitions $R(x)$ into 2^m cells, each containing at most 128ℓ elements. The key observation, stated in Step 1, is that if the procedure does not halt in Step 1 then it is indeed the case that h induces such a partition.

The fact that these cells may contain a different number of elements is immaterial, because each element is output with the same probability (i.e., $1/128\ell$). What matters is that the average number of elements in the cells is sufficiently large, because this average number determines the probability that the procedure outputs an element of $R(x)$ (rather than $\perp$). Specifically, the latter probability equals the aforementioned average number (which equals $|R(x)|/2^m$) divided by 128ℓ. Using $m \leq \max(0, 1+\log_2(2|R(x)|) - 6 - \log_2 \ell)$, we have $|R(x)|/2^m \geq \min(|R(x)|, 16\ell)$, which means that the procedure outputs some element of $R(x)$ with probability at least $\min((|R(x)|/128\ell), (1/8))$. ■

Technical comments. We can easily improve the performance of Construction 6.30 by dealing separately with the case $m = 0$. In such a case, Step 3 can be simplified and improved by uniformly selecting and outputting an element of S_λ (which equals $R(x)$). Under this modification, the procedure outputs some element of $R(x)$ with probability at least 1/8. In any case, recall that the probability that a uniform generation procedure outputs $\perp$ can be deceased by repeated invocations.

Chapter Notes

One key aspect of randomized procedures is their success probability, which is obviously a quantitative notion. This aspect provides a clear connection between probabilistic polynomial-time algorithms considered in Section 6.1 and the counting problems considered in Section 6.2 (see also Exercise 6.17). More appealing connections between randomized procedures and counting problems (e.g., the application of randomization in approximate counting) are presented in Section 6.2. These connections justify the presentation of these two topics in the same chapter.

Randomized algorithms

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 been diminished due to subsequent finding, but the fundamental questions that emerged remain fascinating regardless of the status of these and other appealing examples (see §6.1.1.1).

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 [198] and of Rabin [172], 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 6.4, which only establishes that deciding primality is in $\mathcal{BPP}$, is commonly attributed to M. Blum.) In the late 1980's, Adleman and Huang [2] proved that deciding primality is in $\mathcal{RP}$

(and thus in $\mathcal{ZPP}$). In the early 2000's, Agrawal, Kayal, and Saxena [3] 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 [155], 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 6.10 is due to Aleliunas, Karp, Lipton, Lovász, and Rackoff [5]. Recall that a deterministic log-space algorithm was found twenty-five years later (see Section 5.2.4 or [178]).

Other randomized algorithms. Although randomized algorithms are more abundant in the context of approximation problems (let alone in other computational settings (cf. §6.1.1.1)), 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., [86, Apdx. B.1] or [157, Sec. 12.4&10.2]), and note the prominent role of randomization in computational number theory (see, e.g., [21] or [157, Chap. 14]). For a general textbook on randomized algorithms, we refer the interested reader to [157].

On the general study of $\mathcal{BPP}$. Turning to the general study of $\mathcal{BPP}$, we note that our presentation of Theorem 6.7 follows the proof idea of Lautemann [141]. A different proof technique, which yields a weaker result but found more applications (see, e.g., Theorem 6.25 and [107]), was presented (independently) by Sipser [194].

On the role of promise problems. In addition to their use in the formulation of Theorem 6.7, promise problems allow for establishing time hierarchy theorems (as in §4.2.1.1) for randomized computation (see Exercise 6.13). 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 Chapter 8 and Section D.4. Specifically, as advocated in Chapter 8, 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. In many cases this leads to reducing the number of coin tosses in such implementations, and at times even to a full (but non-trivial) derandomization (see Sections 8.4 and 8.5). A less radical approach is presented in Section D.4, which 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.

Counting problems

The counting class $\#\mathcal{P}$ was introduced by Valiant [215], who proved that computing the permanent of 0/1-matrices is $\#\mathcal{P}$-complete (i.e., Theorem 6.18). Interestingly, like in the case of Cook's introduction of NP-completeness [55], Valiant's motivation was determining the complexity of a specific problem (i.e., the permanent).

Our presentation of Theorem 6.18 is based both on Valiant's paper [215] and on subsequent studies (most notably [29]). Specifically, the high-level structure of the reduction presented in Proposition 6.19 as well as the "structured" design of the clause gadget is taken from [215], whereas the Deus Ex Machina gadget presented in Figure 6.3 is based on [29]. The proof of Proposition 6.20 is also based on [29] (with some variants). Turning back to the design of clause gadgets we regret not being able to cite and/or use a systematic study of this design problem.

As noted in the main text, we decided not to present a proof of Toda's Theorem [207], which asserts that every set in $\mathcal{PH}$ is Cook-reducible to $\#\mathcal{P}$ (i.e., Theorem 6.14). A proof of a related result appears in Section F.1 (implying that $\mathcal{PH}$ is reducible to $\#\mathcal{P}$ via probabilistic polynomial-time reductions). Alternative proofs can be found in [127, 199, 207].

Approximate counting and related problems. The approximation procedure for $\#\mathcal{P}$ is due to Stockmeyer [201], following an idea of Sipser [194]. Our exposition, however, follows further developments in the area. The randomized reduction of $\mathcal{NP}$ to problems of unique solutions was discovered by Valiant and Vazirani [217]. Again, our exposition is a bit different.

The connection between approximate counting and uniform generation (presented in §6.2.4.1) was discovered by Jerrum, Valiant, and Vazirani [125], and turned out to be very useful in the design of algorithms (e.g., in the "Markov Chain approach" (see [157, Sec. 11.3.1])). The direct procedure for uniform generation (presented in §6.2.4.2) is taken from [26].

In continuation to §6.2.2.1, which is based on [130], we refer the interested reader to [124], which presents a probabilistic polynomial-time algorithm for approximating the permanent of non-negative matrices. This fascinating algorithm is based on the fact that knowing (approximately) certain parameters of a non-negative matrix M allows to approximate the same parameters for a matrix M', provided that M and M' are sufficiently similar. Specifically, M and M' may differ only on a single entry, and the ratio of the corresponding values must be sufficiently close to one. Needless to say, the actual observation (is not generic but rather) refers to specific parameters of the matrix, which include its permanent. Thus, given a matrix M for which we need to approximate the permanent, we consider a sequence of matrices $M_0, ..., M_t \approx M$ such that M_0 is the all 1's matrix (for which it is easy to evaluate the said parameters), and each M_{i+1} is obtained from M_i by reducing some adequate entry by a factor sufficiently close to one. This process of (polynomially many) gradual changes, allows to transform the dummy matrix M_0 into a matrix M_t that is very close to M (and hence has a permanent that is very close to the permanent of M). Thus, approximately obtaining the parameters of M_t allows to approximate the permanent of M.

Finally, we note that Section 10.1.1 provides a treatment of a different type of approximation problems. Specifically, when given an instance x (for a search problem R), rather than seeking an approximation of the number of solutions (i.e., $\#R(x)$), one seeks an approximation of the value of the best solution (i.e., best $y \in R(x)$), where the value of a solution is defined by an auxiliary function.

Exercises

Exercise 6.1 Show that if a search (resp., decision) problem can be solved by a probabilistic polynomial-time algorithm having zero failure probability, then the problem can be solve by a deterministic polynomial-time algorithm.
(Hint: replace the internal coin tosses by a fixed outcome that is easy to generate deterministically (e.g., the all-zero sequence).)

Exercise 6.2 (randomized reductions) In continuation to the definitions presented at the beginning of Section 6.1, 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 6.3 (on the definition of probabilistically solving a search problem) In continuation to the discussion at the beginning of Section 6.1.1, 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 6.4 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 $\Pr[A(x)=\perp] > 0.5+(1/p(|x|)))$?

Note that R is not necessarily in $\mathcal{PC}$. Indeed, in the case that $R \in \mathcal{PC}$ we can eliminate the error probability for every $x \notin S_R$, and perform error-reduction as in $\mathcal{RP}$.

Exercise 6.4 (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 6.5 (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 6.6 (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 6.7 (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 6.8 Let $\mathcal{BPP}$ and co$\mathcal{RP}$ be classes of promise problems (as in Theorem 6.7).

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 6.9 (randomness-efficient error-reductions) Note that standard error-reduction (as in Exercise 6.4) yields error probability δ at the cost of increasing the randomness complexity by a *factor* of $O(\log(1/\delta))$. Using the randomness-efficient error-reductions outlined in §D.4.1.3, 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 6.7.

Exercise 6.10 In continuation to the illustrative paragraph in the proof of Theorem 6.7, consider the promise problem $\Pi' = (\Pi'_{\rm yes}, \Pi'_{\rm no})$ such that $\Pi'_{\rm yes} = \{(x, r') : |r'| = p'(|x|) \wedge (\forall r'' \in \{0,1\}^{|r'|})\, A'(x, r'r'') = 1\}$ and $\Pi'_{\rm no} = \{(x, r') : x \notin S\}$. Recall that for every x it holds that $\Pr_{r\in\{0,1\}^{2p'(|x|)}}[A'(x,r) \neq \chi_S(x)] < 2^{-(p'(|x|)+1)}$.

1. Show that mapping x to (x, r'), where r' is uniformly distributed in $\{0,1\}^{p'(|x|)}$, 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 6.11 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 $1 - \exp(-\mathrm{poly}(|x|))$ 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 6.12 (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 $y \in \{0,1\}^{\mathrm{poly}(|x|)}$ 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 $y \in \{0,1\}^{\mathrm{poly}(|x|)}$ 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 6.7, and note that an adequate sequence shifts (to be used by the verifier) can be incorporated in the single message sent by the prover.

Exercise 6.13 (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 Theorem 4.3 and Corollary 4.4.

Guideline: Analogously to the proof of Theorem 4.3, 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 (called admissible) that have two-sided error at most $1/3$ (on every input). This restriction 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 6.14 (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* integer e such that $s^e \equiv 1 \pmod P$, and (once such e is found) we may output $s^{(e+1)/2} \bmod P$. (In Item 1, we used $e = (P-1)/2$, which is odd since $P \equiv 3 \pmod 4$.)

   Show that it suffices to find an *odd* integer e together with a residue t and an *even* integer e' such that $s^e t^{e'} \equiv 1 \pmod P$, because $s \equiv s^{e+1} t^{e'} \equiv (s^{(e+1)/2} t^{e'/2})^2$.

3. Given a prime $P \equiv 1 \pmod 4$, a quadratic residue s, and a quadratic non-residue t (equiv., $t^{(P-1)/2} \equiv -1 \pmod P$), show that e and e' as in Item 2 can be efficiently found.[19]

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

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

Exercise 6.15 (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 6.16 (analysis of random walks on arbitrary undirected graphs) In order to complete the proof of Proposition 6.11, 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,

[19] 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 that for some $i' > i > 0$ and j' it holds that $s^{(2j+1)\cdot 2^{i}} t^{(2j'+1)\cdot 2^{i'}} \equiv 1 \pmod P$, show how to find $i'' > i - 1$ and j'' such that $s^{(2j+1)\cdot 2^{i-1}} t^{(2j''+1)\cdot 2^{i''}} \equiv 1 \pmod P$. (Extra hint: $s^{(2j+1)\cdot 2^{i-1}} t^{(2j'+1)\cdot 2^{i'-1}} \equiv \pm 1 \pmod P$ and $t^{(2j+1)\cdot 2^{i_0}} \equiv -1 \pmod P$.) Thus, starting with $i = i_0$, we reach $i = 1$, at which point we have what we need.

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 $\chi_{v,u}(i) \stackrel{\rm def}{=} 1$ if the edge $\{u,v\}$ was traversed from v to u in the $i^{\rm th}$ step of such a random walk and $\chi_{v,u}(i) \stackrel{\rm def}{=} 0$ 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 6.17 (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 polynomial-time algorithm A such that, for every x, it holds that $x \in S$ if and only if $\mathsf{Pr}[A(x)=1] > 1/2$. 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 as defined in Proposition 6.13. 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 $\mathsf{Pr}[A'(x)=1] > 1/2$.

Exercise 6.18 (artificial $\#\mathcal{P}$-complete problems) Show that there exists a relation $R \in \mathcal{PC}$ such that $\#R$ is $\#\mathcal{P}$-complete and $S_R = \{0,1\}^*$.

Guideline: For any $\#\mathcal{P}$-complete problem R', define $R = \{(x,1y) : (x,y) \in R'\} \cup \{(x,10^{|x|}) : x \in \{0,1\}^*\}$.

Exercise 6.19 (enumeration problems) For any binary relation R, define the enumeration problem of R as a function $f_R : \{0,1\}^* \times \mathbb{N} \to \{0,1\}^* \cup \{\perp\}$ such that $f_R(x,i)$ equals the $i^{\rm th}$ element in $|R(x)|$ if $|R(x)| \geq i$ and $f_R(x,i) = \perp$ otherwise. The above definition refers to the standard lexicographic order on strings, but any other efficient order of strings will do.[20]

1. Prove that, for any polynomially bounded R, computing $\#R$ is reducible to computing f_R.

2. Prove that, for any $R \in \mathcal{PC}$, computing f_R is reducible to some problem in $\#\mathcal{P}$.

[20] An order of strings is a 1-1 and onto mapping μ from the natural numbers to the set of all strings. Such order is called efficient if both μ and its inverse are efficiently computable. The standard lexicographic order satisfies $\mu(i) = y$ if the (compact) binary expansion of i equals $1y$; that is $\mu(1) = \lambda$, $\mu(2) = 0$, $\mu(3) = 1$, $\mu(4) = 00$, etc.

Guideline: Consider the binary relation $R' = \{(\langle x,b\rangle, y) : (x,y) \in R \wedge y \leq b\}$, and show that f_R is reducible to $\#R'$. (Extra hint: Note that $f_R(x,i) = y$ if and only if $|R'(\langle x,y\rangle)| = i$ and for every $y' < y$ it holds that $|R'(\langle x,y'\rangle)| < i$.)

Exercise 6.20 (computing the permanent of integer matrices) Prove that computing the permanent of matrices with 0/1-entries is computationally equivalent to computing the number of perfect matchings in bipartite graphs.
(Hint: Given a bipartite graph $G = ((X,Y),E)$, consider the matrix M representing the edges between X and Y (i.e., the (i,j)-entry in M is 1 if the i^{th} vertex of X is connected to the j^{th} entry of Y), and note that only perfect matchings in G contribute to the permanent of M.)

Exercise 6.21 (computing the permanent modulo 3) Combining Proposition 6.19 and Theorem 6.27, prove that for every integer $n > 1$ that is relatively prime to c, computing the permanent modulo n is NP-hard under randomized reductions.[21] Since Proposition 6.19 holds for $c = 2^{10}$, hardness holds for every odd integer $n > 1$.

Guideline: Apply the reduction of Proposition 6.19 to the promise problem of deciding whether a 3CNF formula has a unique satisfiable assignment or is unsatisfiable. Use the fact that n does not divide any power of c.

Exercise 6.22 (negative values in Proposition 6.19) Assuming $\mathcal{P} \neq \mathcal{NP}$, prove that Proposition 6.19 cannot hold for a set I containing only non-negative integers. Note that the claim holds even if the set I is not finite (and even if I is the set of all non-negative integers).

Guideline: A reduction as in Proposition 6.19 yields a Karp-reduction of `3SAT` to deciding whether the permanent of a matrix with entries in I is non-zero. Note that the permanent of a *non-negative* matrix is non-zero if and only if the corresponding bipartite graph has a perfect matching.

Exercise 6.23 (high-level analysis of the permanent reduction) Establish the correctness of the high-level reduction presented in the proof of Proposition 6.19. That is, show that if the clause gadget satisfies the three conditions postulated in the said proof, then each satisfying assignment of ϕ contributes exactly c^m to the SWCC of G_ϕ whereas unsatisfying assignments have no contribution.

Guideline: Cluster the cycle covers of G_ϕ according to the set of track edges that they use (i.e., the edges of the cycle cover that belong to the various tracks). (Note the correspondence between these edges and the external edges used in the definition of the gadget's properties.) Using the postulated conditions (regarding the clause gadget) prove that, for each such set T of track edges, if the sum of the weights of all cycle covers that use the track edges T is non-zero then the following hold:

1. The intersection of T with the set of track edges incident at each specific clause gadget is non-empty. Furthermore, if this set contains an incoming edge (resp.,

[21] Actually, a sufficient condition is that n does not divide any power of c. Thus (referring to $c = 2^{10}$), hardness holds for every integer $n > 1$ that is not a power of 2. On the other hand, for any fixed $n = 2^e$, the permanent modulo n can be computed in polynomial-time [215, Thm. 3].

outgoing edge) of some entry-vertex (resp., exit-vertex) then it also contains an outgoing edge (resp., incoming edge) of the corresponding exit-vertex (resp., entry-vertex).

2. If T contains an edge that belongs to some track then it contains all edges of this track. It follows that, for each variable x, the set T contains the edges of a single track associated with x.
3. The tracks "picked" by T correspond to a single truth assignment to the variables of ϕ, and this assignment satisfies ϕ (because, for each clause, T contains an external edge that corresponds to a literal that satisfies this clause).

It follows that each satisfying assignment of ϕ contributes exactly c^m to the SWCC of G_ϕ.

Exercise 6.24 (analysis of the implementation of the clause gadget) Establish the correctness of the implementation of the clause gadget presented in the proof of Proposition 6.19. That is, show that if the box satisfy the three conditions postulated in the said proof, then the clause gadget of Figure 6.4 satisfies the conditions postulated for it.

Guideline: Cluster the cycle covers of a gadget according to the set of non-box edges that they use, where non-box edges are the edges shown in Figure 6.4. Using the postulated conditions (regarding the box) prove that, for each set S of non-box edges, if the sum of the weights of all cycle covers that use the non-box edges S is non-zero then the following hold:

1. The intersection of S with the set of edges incident at each box must contain two (non-selfloop) edges, one incident at each of the box's terminals. Needless to say, one edge is incoming and the other outgoing. Referring to the six edges that connects one of the six designated vertices (of the gadget) with the corresponding box terminals as connectives, note that if S contains a connective incident at the terminal of some box then it must also contain the connective incident at the other terminal. In such a case, we say that this box is picked by S,
2. Each of the three (literal-designated) boxes that is not picked by S is "traversed" from left to right (i.e., the cycle cover contains an incoming edge of the left terminal and an outgoing edge of the right terminal). Thus, the set S must contain a connective, because otherwise no directed cycle may cover the leftmost vertex shown in Figure 6.4. That is, S must pick some box.
3. The set S is fully determined by the non-empty set of boxes that it picks.

The postulated properties of the clause gadget follow, with $c = b^5$.

Exercise 6.25 (analysis of the design of a box for the clause gadget) Prove that the 4-by-4 matrix presented in Eq. (6.4) satisfies the properties postulated for the "box" used in the second part of the proof of Proposition 6.19. In particular:

1. Show a correspondence between the conditions required of the box and conditions regarding the value of the permanent of certain sub-matrices of the adjacency matrix of the graph.

   (Hint: For example, show that the first condition correspond to requiring that the value of the permanent of the entire matrix equals zero. The second condition refers to sub-matrices obtained by omitting either the first row and fourth column or the fourth row and first column.)

2. Verify that the matrix in Eq. (6.4) satisfies the aforementioned conditions (regarding the value of the permanent of certain sub-matrices).

Prove that no 3-by-3 matrix (and thus also no 2-by-2 matrix) can satisfy the aforementioned conditions.

Exercise 6.26 (error reduction for approximate counting) Show that the error probability δ in Definition 6.22 can be reduced from $1/3$ (or even $(1/2) + (1/\text{poly}(|x|))$ to $\exp(-\text{poly}(|x|))$.

Guideline: Invoke the weaker procedure for an adequate number of times and take the *median* value among the values obtained in these invocations.

Exercise 6.27 (strong approximation for some $\#\mathcal{P}$-complete problems) Show that there exists $\#\mathcal{P}$-complete problems (albeit unnatural ones) for which an $(\varepsilon, 0)$-approximation can be found by a (deterministic) polynomial-time algorithm. Furthermore, the running-time depends polynomially on $1/\varepsilon$.

Guideline: Combine any $\#\mathcal{P}$-complete problem referring to some $R_1 \in \mathcal{PC}$ with a trivial counting problem (e.g., such as the counting problem associated with $R_2 = \cup_{n\in\mathbb{N}}\{(x,y) : x,y \in \{0,1\}^n\}$). Show that, without loss of generality, that $(x,y) \in R_1$ implies $|x| = |y|$ and that $\#R_1(x) \leq 2^{|x|/2}$. Prove that the counting problem of $R = \{(x,1y) : (x,y) \in R_1\} \cup \{(x,0y) : (x,y) \in R_2\}$ is $\#\mathcal{P}$-complete. Present a deterministic algorithm that, on input x and $\varepsilon > 0$, outputs an $(\varepsilon, 0)$-approximation of $\#R(x)$ in time $\text{poly}(|x|/\varepsilon)$.

Exercise 6.28 (relative approximation for DNF satisfaction) Referring to the text of §6.2.2.1, prove the following claims.

1. Both assumptions regarding the general setting hold in case $S_i = C_i^{-1}(1)$, where $C_i^{-1}(1)$ denotes the set of truth assignments that satisfy the conjunction C_i.

   Guideline: In establishing the second assumption note that it reduces to the conjunction of the following two assumptions:

   (a) Given i, one can efficiently generate a uniformly distributed element of S_i. Actually, generating a distribution that is almost uniform over S_i suffices.

   (b) Given i and x, one can efficiently determine whether $x \in S_i$.

2. Prove Proposition 6.24, relating to details such as the error probability in an implementation of Construction 6.23.

3. Note that Construction 6.23 does not require exact computation of $|S_i|$. Analyze the output distribution in the case that we can only approximate $|S_i|$ up-to a factor of $1 \pm \varepsilon'$.

Exercise 6.29 (reducing the relative deviation in approximate counting) Prove that, for any $R \in \mathcal{PC}$ and every polynomial p and constant $\delta < 0.5$, there exists $R' \in \mathcal{PC}$ such that $(1/p, \delta)$-approximation for $\#R$ is reducible to $(1/2, \delta)$-approximation for $\#R'$.

Guideline: For $t(n) = \Theta(p(n))$, let $R' = \{(x,(y_1,...,y_{t(|x|)})) : (\forall i)\,(x,y_i) \in R\}$. Note that $|R(x)| = |R'(x)|^{1/t(|x|)}$, and thus if $a = (1 \pm (1/2)) \cdot |R'(x)|$ then $a^{1/t(|x|)} = (1 \pm (1/2))^{1/t(|x|)} \cdot |R(x)|$.
Furthermore, for any $F(n) = \exp(\text{poly}(n))$, prove that there exists $R'' \in \mathcal{PC}$ such that $(1/p,\delta)$-approximation for $\#R$ is reducible to approximating $\#R''$ to within a factor of F with error probability δ.
(Hint: Same as the main part (using $t(n) = \Theta(p(n) \cdot \log F(n))$).)

Exercise 6.30 (deviation reduction in approximate counting, cont.) In continuation to Exercise 6.29, prove that if R is NP-complete via parsimonious reductions then, for every positive polynomial p and constant $\delta < 0.5$, the problem of $(1/p,\delta)$-approximation for $\#R$ is reducible to $(1/2,\delta)$-approximation for $\#R$.
(Hint: Compose the reduction (to the problem of $(1/2,\delta)$-approximation for $\#R'$) provided in Exercise 6.29 with the parsimonious reduction of $\#R'$ to $\#R$.)
Prove that, for every function F' such that $F'(n) = \exp(n^{o(1)})$, we can also reduce the aforementioned problems to the problem of approximating $\#R$ to within a factor of F' with error probability δ.

Guideline: Using R'' as in Exercise 6.29, we encounter a technical difficulty. The issue is that the composition of the ("amplifying") reduction of $\#R$ to $\#R''$ with the parsimonious reduction of $\#R''$ to $\#R$ may increase the length of the instance. Indeed, the length of the new instance is polynomial in the length of the original instance, but this polynomial may depend on R'', which in turn depends on F'. Thus, we cannot use $F'(n) = \exp(n^{1/O(1)})$ but $F'(n) = \exp(n^{o(1)})$ is fine.

Exercise 6.31 Referring to the procedure in the proof Theorem 6.25, show how to use an NP-oracle in order to determine whether the number of solutions that "pass a random sieve" is greater than t. You are allowed queries of length polynomial in the length of x, h and in the size of t.
(Hint: Consider the set $S'_{R,H} \stackrel{\text{def}}{=} \{(x,i,h,1^t) : \exists y_1,...,y_t \text{ s.t. } \psi'(x,h,y_1,...,y_t)\}$, where $\psi'(x,h,y_1,...,y_t)$ holds if and only if the y_j are different and for every j it holds that $(x,y_j) \in R \wedge h(y_j) = 0^i$.)

Exercise 6.32 (parsimonious reductions and Theorem 6.27) Demonstrate the importance of parsimonious reductions in Theorem 6.27 by proving the following:

1. There exists a search problem $R \in \mathcal{PC}$ such that every problem in $\mathcal{PC}$ is reducible to R (by a non-parsimonious reduction) and still the the promise problem $(\texttt{US}_R, \overline{S}_R)$ is decidable in polynomial-time.

   Guideline: Consider the following artificial witness relation R for $\texttt{SAT}$ in which $(\phi, \sigma\tau) \in R$ if $\sigma \in \{0,1\}$ and τ satisfies ϕ. Note that the standard witness relation of $\texttt{SAT}$ is reducible to R, but this reduction is not parsimonious. Also note that $\texttt{US}_R = \emptyset$ and thus $(\texttt{US}_R, \overline{S}_R)$ is trivial.

2. There exists a search problem $R \in \mathcal{PC}$ such that $\#R$ is $\#\mathcal{P}$-complete and still the the promise problem $(\texttt{US}_R, \overline{S}_R)$ is decidable in polynomial-time.

   Guideline: Just use the relation suggested in the guideline to Part 1. An alternative proof relies on Theorem 6.18 and on the fact that it is easy to decide

$(\mathtt{US}_R, \overline{S}_R)$ when R is the corresponding perfect matching relation (by computing the determinant).

Exercise 6.33 Prove that SAT is randomly reducible to deciding unique solution for SAT, *without using the fact that* SAT *is NP-complete via parsimonious reductions.*

Guideline: Follow the proof of Theorem 6.27, while using the family of pairwise independent hashing functions provided in Construction D.3 (or in Eq. (8.18)). Note that, in this case, the condition $(\tau \in R_{\mathtt{SAT}}(\phi)) \wedge (h(\tau) = 0^i)$ can be directly encoded as a CNF formula. That is, consider the formula ϕ_h such that $\phi_h(z) \stackrel{\rm def}{=} \phi(z) \wedge (h(z) = 0^i)$, and note that $h(z) = 0^i$ can be written as the conjunction of i clauses, where each clause is a CNF that is logically equivalent to the parity of some of the bits of z (where the identity of these bits is determined by h).

Exercise 6.34 (an alternative procedure for approximate counting) Adapt Step 1 of Construction 6.30 so to obtain an approximate counting procedure for $\#R$.

Guideline: For $m = 0, 1, ...\ell$, the procedure invokes Step 1 of Construction 6.30 until a negative answer is obtained, and outputs 2^m for the current value of m. For $|R(x)| > 128\ell$, this yields a constant factor approximation of $|R(x)|$. In fact, we can obtain a better estimate by making additional queries at iteration m (i.e., queries of the form $(x, h, 1^i)$ for $i = 16\ell, ..., 128\ell$). The case $|R(x)| \leq 128\ell$ can be treated by using Step 2 of Construction 6.30, in which case we obtain an exact count.

Exercise 6.35 Let R be an arbitrary $\mathcal{PC}$-complete search problem. Show that approximate counting and uniform generation for R can be randomly reduced to deciding membership in S_R, where by approximate counting we mean a $(1-(1/p)$-approximation for any polynomial p.

Guideline: Note that Construction 6.30 yields such procedures (see also Exercise 6.34), except that they make oracle calls to some other set in $\mathcal{NP}$. Using the NP-completeness of S_R, we are done.

Chapter 10

Relaxing the Requirements

The philosophers have only interpreted the world, in various ways; the point is to change it.

Karl Marx, Theses on Feuerbach

In light of the apparent infeasibility of solving numerous natural computational problems, it is natural to ask whether these problems can be relaxed in a way that is both useful for applications and allows for feasible solving procedures. We stress two aspects about the foregoing question: on one hand, the relaxation should be sufficiently good for the intended applications; but, on the other hand, it should be significantly different from the original formulation of the problem so to escape the infeasibility of the latter. We note that whether a relaxation is adequate for an intended application depends on the application, and thus much of the material in this chapter is less robust (or generic) than the treatment of the non-relaxed computational problems.

Summary: We consider two types of relaxations. The first type of relaxation refers to the computational problems themselves; that is, for each problem instance we *extend the set of admissible solutions*. In the context of search problems this means settling for solutions that have a value that is "sufficiently close" to the value of the optimal solution (with respect to some value function). Needless to say, the specific meaning of 'sufficiently close' is part of the definition of the relaxed problem. In the context of decision problems this means that for some instances both answers are considered valid; put differently, we consider promise problems in which the no-instances are "far" from the yes-instances in some adequate sense (which is part of the definition of the relaxed problem).

The second type of relaxation deviates from the requirement that the solver provides an adequate answer on each valid instance. Instead, the behavior of the solver is analyzed with respect to a predetermined

input distribution (or a class of such distributions), and bad behavior may occur with negligible probability where the probability is taken over this input distribution. That is, we replace worst-case analysis by *average-case* (or rather *typical-case*) *analysis.* Needless to say, a major component in this approach is limiting the class of distributions in a way that, on one hand, allows for various types of natural distributions and, on the other hand, prevents the collapse of the corresponding notion of average-case complexity to the standard worst-case complexity.

10.1 Approximation

The notion of approximation is a natural one, and has arisen also in other disciplines. Approximation is most commonly used in references to quantities (e.g., "the length of one meter is approximately forty inches"), but it is also used when referring to qualities (e.g., "an approximately correct account of a historical event"). In the context of computation, the notion of approximation modifies computational tasks such as search and decision problems. (In fact, we have already encountered it as a modifier of counting problems; see Section 6.2.2.)

Two major questions regarding approximation are (1) what is a "good" approximation, and (2) can it be found easier than finding an exact solution. The answer to the first question seems intimately related to the specific computational task at hand and to its role in the wider context (i.e., the higher level application): a good approximation is one that suffices for the intended application. Indeed, the importance of certain approximation problems is much more subjective than the importance of the corresponding optimization problems. This fact seems to stand in the way of attempts at providing a *comprehensive* theory of *natural* approximation problems (e.g., general classes of natural approximation problems that are shown to be computationally equivalent).

Turning to the second question, we note that in numerous cases natural approximation problems seem to be significantly easier than the corresponding original ("exact") problems. On the other hand, in numerous other cases, natural approximation problems are computationally equivalent to the original problems. We shall exemplify both cases by reviewing some specific results, but regret not being able to provide any systematic classification.

Mimicking the two standard uses of the word *approximation*, we shall distinguish between approximation problems that are of the "search type" and problems that are have a clear "decisional" flavor. In the first case we shall refer to a function that assigns values to possible solutions (of a search problem); whereas in the second case we shall refer to distances between instances (of a decision problem). Needless to say, at times the same computational problem may be cast in both ways, but for most natural approximation problems one of the two frameworks is more appealing than the other.

The common theme is that in both cases we extend the set of admissible solutions. In the case of search problems, we extend the set of optimal solutions by including also almost-optimal solutions. In the case of decision problems, we extend the set of solutions by allowing an arbitrary answer (solution) to some instances, which may be viewed as a promise problem that disallows these instances. In this case we focus on promise problems in which the yes and no-instances are far apart (and the instances that violate the promise are closed to yes-instances).

Teaching note: Most of the results presented in this section refer to specific computational problems and (with one exception) are presented without a proof. In view of the complexity of the corresponding proofs and the merely illustrative role of these results in the context of complexity theory, we recommend doing the same in class.

10.1.1 Search or Optimization

As noted in Section 2.2.2, many search problems involve a set of potential solutions (per each problem instance) such that different solutions are assigned different "values" (resp., "costs") by some "value" (resp., "cost") function. In such a case, one is interested in finding a solution of maximum value (resp., minimum cost). A corresponding approximation problem may refer to finding a solution of approximately maximum value (resp., approximately minimum cost), where the specification of the desired level of approximation is part of the problem's definition. Let us elaborate.

For concreteness, we focus on the case of a value that we wish to maximize. For greater flexibility, we allow the value of the solution to depend also on the instance itself. Thus, for a (polynomially bounded) binary relation R and a *value function* $f:\{0,1\}^*\times\{0,1\}^*\to\mathbb{R}$, we consider the problem of finding solutions (with respect to R) that maximize the value of f. That is, given x (such that $R(x)\neq\emptyset$), the task is finding $y\in R(x)$ such that $f(x,y)=v_x$, where v_x is the maximum value of $f(x,y')$ over all $y'\in R(x)$. Typically, R is in $\mathcal{PC}$ and f is polynomial-time computable.[1] Indeed, without loss of generality, we may assume that for every x it holds that $R(x)=\{0,1\}^{\ell(|x|)}$ for some polynomial ℓ (see Exercise 2.8). Thus, the optimization problem is recast as the following search problem: *given x, find y such that $f(x,y)=v_x$, where $v_x=\max_{y'\in\{0,1\}^{\ell(|x|)}}\{f(x,y')\}$.*

We shall focus on *relative* approximation problems, where for some gap function $g:\{0,1\}^*\to\{r\in\mathbb{R}:r\geq 1\}$ the (maximization) task is finding y such that $f(x,y)\geq v_x/g(x)$. Indeed, in some cases the approximation factor is stated as a function of the length of the input (i.e., $g(x)=g'(|x|)$ for some $g':\mathbb{N}\to\{r\in\mathbb{R}:r\geq 1\}$), but often the approximation factor is stated in terms of some more refined parameter of the input (e.g., as a function of the number of vertices in a graph). Typically, g is polynomial-time computable.

Definition 10.1 (g-factor approximation): *Let $f:\{0,1\}^*\times\{0,1\}^*\to\mathbb{R}$, $\ell:\mathbb{N}\to\mathbb{N}$, and $g:\{0,1\}^*\to\{r\in\mathbb{R}:r\geq 1\}$.*

[1] In this case, we may assume without loss of generality that the function f depends only on the solution. This can be obtained by redefining the relation R such that each solution $y\in R(x)$ consists of a pair of the form (x,y'). Needless to say, this modification cannot be applied along with getting rid of R (as in Exercise 2.8).

Maximization version: *The* g-factor approximation of maximizing f (w.r.t ℓ) *is the search problem* R *such that* $R(x) = \{y \in \{0,1\}^{\ell(|x|)} : f(x,y) \geq v_x/g(x)\}$, *where* $v_x = \max_{y' \in \{0,1\}^{\ell(|x|)}}\{f(x,y')\}$.

Minimization version: *The* g-factor approximation of minimizing f (w.r.t ℓ) *is the search problem* R *such that* $R(x) = \{y \in \{0,1\}^{\ell(|x|)} : f(x,y) \leq g(x) \cdot c_x\}$, *where* $c_x = \min_{y' \in \{0,1\}^{\ell(|x|)}}\{f(x,y')\}$.

We note that for numerous NP-complete optimization problems polynomial-time algorithms provide meaningful approximations. A few examples will be mentioned in §10.1.1.1. In contrast, for numerous other NP-complete optimization problems, natural approximation problems are computationally equivalent to the corresponding optimization problem. A few examples will be mentioned in §10.1.1.2, where we also introduce the notion of a *gap problem*, which is a promise problem (of the decision type) intended to capture the difficulty of the (approximate) search problem.

10.1.1.1 A few positive examples

Let us start with a trivial example. Considering a problem such as finding the maximum clique in a graph, we note that finding a linear factor approximation is trivial (i.e., given a graph $G = (V,E)$, we may output any vertex in V as a $|V|$-factor approximation of the maximum clique in G). A famous non-trivial example is presented next.

Proposition 10.2 (factor two approximation to `minimum Vertex Cover`): *There exists a polynomial-time approximation algorithm that given a graph $G = (V,E)$ outputs a vertex cover that is at most twice as large as the minimum vertex cover of G.*

We warn that an approximation algorithm for `minimum Vertex Cover` does not yield such an algorithm for the complementary problem (of `maximum Independent Set`). This phenomenon stands in contrast to the case of optimization, where an optimal solution for one problem (e.g., `minimum Vertex Cover`) yields an optimal solution for the complementary problem (`maximum Independent Set`).

Proof Sketch: The main observation is a connection between the set of maximal matchings and the set of vertex covers in a graph. Let M be any *maximal* matching in the graph $G = (V,E)$; that is, $M \subseteq E$ is a matching but augmenting it by any single edge yields a set that is not a matching. Then, on one hand, the set of all vertices participating in M is a vertex cover of G, and, on the other hand, each vertex cover of G must contain at least one vertex of each edge of M. Thus, we can find the desired vertex cover by finding a maximal matching, which in turn can be found by a greedy algorithm. □

Another example. An instance of the traveling salesman problem (TSP) consists of a symmetric matrix of distances between pairs of points, and the task is finding a shortest tour that passes through all points. In general, no reasonable approximation is feasible for this problem (see Exercise 10.1), but here we consider two special cases in which the distances satisfies some natural constraints (and pretty good approximations are feasible).

Theorem 10.3 (approximations to special cases of TSP): *Polynomial-time algorithms exists for the following two computational problems.*

1. *Providing a 1.5-factor approximation for the special case of TSP in which the distances satisfy the triangle inequality.*

2. *For every $\varepsilon > 1$, providing a $(1+\varepsilon)$-factor approximation for the special case of Euclidean TSP* (i.e., for some constant k (e.g., $k = 2$), the points reside in a k-dimensional Euclidean space, and the distances refer to the standard Euclidean norm).

A weaker version of Part 1 is given in Exercise 10.2. A detailed survey of Part 2 is provided in [12]. We note the difference examplified by the two items of Theorem 10.3: Whereas Part 1 provides a polynomial-time approximation for a specific constant factor, Part 2 provides such an algorithm for any constant factor. Such a result is called a *polynomial-time approximation scheme* (abbrev. PTAS).

10.1.1.2 A few negative examples

Let us start again with a trivial example. Considering a problem such as finding the maximum clique in a graph, we note that given a graph $G = (V, E)$ finding a $(1 + |V|^{-1})$-factor approximation of the maximum clique in G is as hard as finding a maximum clique in G. Indeed, this "result" is not really meaningful. In contrast, building on the PCP Theorem (Theorem 9.16), one may prove that finding a $|V|^{1-o(1)}$-factor approximation of the maximum clique in G is as hard as finding a maximum clique in G. This follows from the fact that the approximation problem is NP-hard (cf. Theorem 10.5).

The statement of inapproximability results is made stronger by referring to a promise problem that consists of distinguishing instances of sufficiently far apart values. Such promise problems are called gap problems, and are typically stated with respect to two bounding functions $g_1, g_2 : \{0,1\}^* \to \mathbb{R}$ (which replace the gap function g of Definition 10.1). Typically, g_1 and g_2 are polynomial-time computable.

Definition 10.4 (gap problem for approximation of f): *Let f be as in Definition 10.1 and $g_1, g_2 : \{0,1\}^* \to \mathbb{R}$.*

Maximization version: *For $g_1 \geq g_2$, the* gap_{g_1,g_2} problem of maximizing f *consists of distinguishing between $\{x : v_x \geq g_1(x)\}$ and $\{x : v_x < g_2(x)\}$, where $v_x = \max_{y \in \{0,1\}^{\ell(|x|)}}\{f(x,y)\}$.*

Minimization version: *For* $g_1 \leq g_2$, *the* gap_{g_1,g_2} problem of minimizing f *consists of distinguishing between* $\{x : c_x \leq g_1(x)\}$ *and* $\{x : c_x > g_2(x)\}$, *where* $c_x = \min_{y \in \{0,1\}^{\ell(|x|)}} \{f(x,y)\}$.

For example, the gap_{g_1,g_2} problem of maximizing the size of a clique in a graph consists of distinguishing between graphs G that have a clique of size $g_1(G)$ and graphs G that have no clique of size $g_2(G)$. In this case, we typically let $g_i(G)$ be a function of the number of vertices in $G = (V,E)$; that is, $g_i(G) = g'_i(|V|)$. Indeed, letting $\omega(G)$ denote the size of the largest clique in the graph G, we let $\texttt{gapClique}_{L,s}$ denote the gap problem of distinguishing between $\{G = (V,E) : \omega(G) \geq L(|V|)\}$ and $\{G = (V,E) : \omega(G) < s(|V|)\}$, where $L \geq s$. Using this terminology, we restate (and strengthen) the aforementioned $|V|^{1-o(1)}$-factor inapproximation of the maximum clique problem.

Theorem 10.5 *For some* $L(N) = N^{1-o(1)}$ *and* $s(N) = N^{o(1)}$, *it holds that* $\texttt{gapClique}_{L,s}$ *is NP-hard.*

The proof of Theorem 10.5 is based on a major refinement of Theorem 9.16 that refers to a PCP system of amortized free-bit complexity that tends to zero (cf. §9.3.4.1). A weaker result, which follows from Theorem 9.16 itself, is presented in Exercise 10.3.

As we shall show next, results of the type of Theorem 10.5 imply the hardness of a corresponding approximation problem; that is, the hardness of deciding a gap problem implies the hardness of a search problem that refers to an analogous factor of approximation.

Proposition 10.6 *Let* f, g_1, g_2 *be as in Definition 10.4 and suppose that these functions are polynomial-time computable. Then the* gap_{g_1,g_2} *problem of maximizing* f (resp., minimizing f) *is reducible to the* g_1/g_2*-factor* (resp., g_2/g_1-factor) *approximation of maximizing* f (resp., minimizing f).

Note that a reduction in the opposite direction does not necessarily exist (even in the case that the underlying optimization problem is self-reducible in some natural sense). Indeed, this is another difference between the current context (of approximation) and the context of optimization problems, where the search problem is reducible to a related decision problem.

Proof Sketch: We focus on the maximization version. On input x, we solve the gap_{g_1,g_2} problem, by making the query x, obtaining the answer y, and ruling that x has value exceeding $g_1(x)$ if and only if $f(x,y) \geq g_2(x)$. Recall that we need to analyze this reduction only on inputs that satisfy the promise. Thus, if $v_x \geq g_1(x)$ then the oracle must return a solution y that satisfies $f(x,y) \geq v_x/(g_1(x)/g_2(x))$, which implies that $f(x,y) \geq g_2(x)$. On the other hand, if $v_x < g_2(x)$ then $f(x,y) \leq v_x < g_2(x)$ holds for any possible solution y. $\square$

Additional examples. Let us consider $\texttt{gapVC}_{s,L}$, the gap_{g_s,g_L} problem of minimizing the vertex cover of a graph, where s and L are constants and $g_s(G) = s \cdot |V|$ (resp., $g_L(G) = L \cdot |V|$) for any graph $G = (V, E)$. Then, Proposition 10.2 implies (via Proposition 10.6) that, for every constant s, the problem $\texttt{gapVC}_{s,2s}$ is solvable in polynomial-time. In contrast, sufficiently narrowing the gap between the two thresholds yields an inapproximability result. In particular:

Theorem 10.7 *For some constants* $0 < s < L < 1$ (e.g., $s = 0.62$ and $L = 0.84$ will do)*, the problem* $\texttt{gapVC}_{s,L}$ *is NP-hard.*

The proof of Theorem 10.7 is based on a complicated refinement of Theorem 9.16. Again, a weaker result follows from Theorem 9.16 itself (see Exercise 10.4).

As noted, refinements of the PCP Theorem (Theorem 9.16) play a key role in establishing inapproximability results such as Theorems 10.5 and 10.7. In that respect, it is adequate to recall that Theorem 9.21 establishes the equivalence of the PCP Theorem itself and the NP-hardness of a gap problem concerning the maximization of the number of clauses that are satisfies in a given 3-CNF formula. Specifically, $\texttt{gapSAT}^3_\varepsilon$ was defined (in Definition 9.20) as the gap problem consisting of distinguishing between satisfiable 3-CNF formulae and 3-CNF formulae for which each truth assignment violates at least an ε fraction of the clauses. Although Theorem 9.21 does not specify the quantitative relation that underlies its qualitative assertion, when (refined and) combined with the best known PCP construction, it does yield the best possible bound.

Theorem 10.8 *For every* $v < 1/8$*, the problem* $\texttt{gapSAT}^3_v$ *is NP-hard.*

On the other hand, $\texttt{gapSAT}^3_{1/8}$ is solvable in polynomial-time.

Sharp threshold. The aforementioned conflicting results (regarding $\texttt{gapSAT}^3_v$) exemplify a sharp threshold on the (factor of) approximation that can be obtained by an efficient algorithm. Another appealing example refers to the following maximization problem in which the instances are systems of linear equations over GF(2) and the task is finding an assignment that satisfies as many equations as possible. Note that by merely selecting an assignment at random, we expect to satisfy half of the equations. Also note that it is easy to determine whether there exists an assignment that satisfies all equations. Let $\texttt{gapLin}_{L,s}$ denote the problem of distinguishing between systems in which one can satisfy at least an L fraction of the equations and systems in which one cannot satisfy an s fraction (or more) of the equations. Then, as just noted, $\texttt{gapLin}_{L,0.5}$ is trivial and $\texttt{gapLin}_{1,s}$ is feasible (for every $s < 1$). In contrast, moving both thresholds (slightly) away from the corresponding extremes, yields an NP-hard gap problem:

Theorem 10.9 *For every constant* $\varepsilon > 0$*, the problem* $\texttt{gapLin}_{1-\varepsilon,0.5+\varepsilon}$ *is NP-hard.*

The proof of Theorem 10.9 is based on a major refinement of Theorem 9.16. In fact, the corresponding PCP system (for NP) is merely a reformulation of Theorem 10.9: the verifier makes three queries and tests a linear condition regarding the answers, while using a logarithmic number of coin tosses. This verifier accepts any yes-instance with probability at least $1-\varepsilon$ (when given oracle access to a suitable proof), and rejects any no-instance with probability at least $0.5-\varepsilon$ (regardless of the oracle being accessed). A weaker result, which follows from Theorem 9.16 itself, is presented in Exercise 10.5.

Gap location. Theorems 10.8 and 10.9 illustrate two opposite situations with respect to the "location" of the "gap" for which the corresponding promise problem is hard. Recall that both gapSAT and gapLin are formulated with respect to two thresholds, where each threshold bounds the fraction of "local" conditions (i.e., clauses or equations) that are satisfiable in the case of yes and no-instances, respectively. In the case of gapSAT the high threshold (referring to yes-instances) was set to 1, and thus only the low threshold (referring to no-instances) remained a free parameter. Nevertheless, a hardness result was established for gapSAT, and furthermore this was achieved for an optimal value of the low threshold (cf. the foregoing discussion of sharp threshold). In contrast, in the case of gapLin setting the high threshold to 1 makes the gap problem efficiently solvable. Thus, the hardness of gapLin was established at a different location of the high threshold. Specifically, hardness (for an optimal value of the ratio of thresholds) was established when setting the high threshold to $1-\varepsilon$, for any $\varepsilon > 0$.

A final comment. All the aforementioned inapproximability results refer to approximation (resp., gap) problems that are relaxations of optimization problems in NP (i.e., the optimization problem is computational equivalent to a decision problem in $\mathcal{NP}$; see Section 2.2.2). In these cases, the NP-hardness of the approximation (resp., gap) problem implies that the corresponding optimization problem is reducible to the approximation (resp., gap) problem. In other words, in these cases nothing is gained by relaxing the original optimization problem, because the relaxed version remains just as hard.

10.1.2 Decision or Property Testing

A natural notion of relaxation for decision problems arises when considering the distance between instances, where a natural notion of distance is the Hamming distance (i.e., the fraction of bits on which two strings disagree). Loosely speaking, this relaxation (called *property testing*) refers to distinguishing inputs that reside in a predetermined set S from inputs that are "relatively far" from any input that resides in the set. Two natural types of promise problems emerge (with respect to any predetermined set S (and the Hamming distance between strings)):

1. *Relaxed decision w.r.t a fixed distance*: Fixing a distance parameter δ, we consider the problem of distinguishing inputs in S from inputs in $\Gamma_\delta(S)$, where
$$\Gamma_\delta(S) \stackrel{\rm def}{=} \{x : \forall z \in S \cap \{0,1\}^{|x|}\ \ \Delta(x,z) > \delta \cdot |x|\} \qquad (10.1)$$
and $\Delta(x_1 \cdots x_m, z_1 \cdots z_m) = |\{i : x_i \neq z_i\}|$ denotes the number of bits on which $x = x_1 \cdots x_m$ and $z = z_1 \cdots z_m$ disagree. Thus, here we consider a

promise problem that is a restriction (or a special case) of the problem of deciding membership in S.

2. *Relaxed decision w.r.t a variable distance*: Here the instances are pairs (x, δ), where x is as in Type 1 and $\delta \in [0,1]$ is a distance parameter. The yes-instances are pairs (x, δ) such that $x \in S$, whereas (x, δ) is a no-instance if $x \in \Gamma_\delta(S)$.

We shall focus on Type 1 formulation, which seems to capture the essential question of whether or not these relaxations lower the complexity of the original decision problem. The study of Type 2 formulation refers to a relatively secondary question, which assumes a positive answer to the first question; that is, assuming that the relaxed form is easier than the original form, we ask how is the complexity of the problem affected by making the distance parameter smaller (which means making the relaxed problem "tighter" and ultimately equivalent to the original problem).

We note that for numerous NP-complete problems there exist natural (Type 1) relaxations that are solvable in polynomial-time. Actually, these algorithms run in *sub-linear* time (specifically polylogarithmic time), when given direct access to the input. A few examples will be presented in §10.1.2.2. As indicated in §10.1.2.2, this is not a generic phenomenon. But before turning to these results, we discuss several important definitional issues.

10.1.2.1 Definitional issues

Property testing is concerned not only with solving relaxed versions of NP-hard problems, but rather solving these problems (as well as problems in $\mathcal{P}$) in *sub-linear time*. Needless to say, such results assume a model of computation in which algorithms have direct access to bits in the (representation of the) input (see Definition 10.10).

Definition 10.10 (a direct access model – conventions): *An algorithm with* direct access to its input *is given its* main input *on a special input device that is accessed as an oracle* (see §1.2.3.5). *In addition, the algorithm is given the length of the input and possibly other parameters on an* secondary input device. *The complexity of such an algorithm is stated in terms of the length of its main input.*

Indeed, the description in §5.2.4.2 refers to such a model, but there the main input is viewed as an oracle and the secondary input is viewed as the input.

Definition 10.11 (property testing for S): *For any fixed $\delta > 0$, the promise problem of distinguishing S from $\Gamma_\delta(S)$ is called* property testing for S (with respect to δ).

Recall that we say that a randomized algorithm solves a promise problem if it accepts every yes-instance (resp., rejects every no-instance) with probability at least 2/3. Thus, a (randomized) property testing for S accepts every input in S (resp., rejects every input in $\Gamma_\delta(S)$) with probability at least 2/3.

The question of representation. The specific representation of the input is of major concern in the current context. This is due to (1) the *effect of the representation on the distance measure* and to (2) the *dependence of direct access machines on the specific representation of the input.* Let us elaborate on both aspects.

1. Recall that we defined the distance between objects in terms of the Hamming distance between their representations. Clearly, in such a case, the choice of representation is crucial and different representations may yield different distance measures. Furthermore, in this case, the distance between objects is not preserved under various (natural) representations that are considered "equivalent" in standard studies of computational complexity. For example, in previous parts of this book, when referring to computational problems concerning graphs, we did not care whether the graphs were represented by their adjacency matrix or by their incidence-lists. In contrast, these two representations induce very different distance measures and correspondingly different property testing problems (see §10.1.2.2). Likewise, the use of padding (and other trivial syntactic conventions) becomes problematic (e.g., when using a significant amount of padding, all objects are deemed close to one another (and property testing for any set becomes trivial)).

2. Since our focus is on sub-linear time algorithms, we may not afford transforming the input from one natural format to another. Thus, representations that are considered equivalent with respect to polynomial-time algorithms, may not be equivalent with respect to sub-linear time algorithms that have a direct access to the representation of the object. For example, adjacency queries and incidence queries cannot emulate one another in small time (i.e., in time that is sub-linear in the number of vertices).

Both aspects are further clarified by the examples provided in §10.1.2.2.

The essential role of the promise. Recall that, for a fixed constant $\delta > 0$, we consider the promise problem of distinguishing S from $\Gamma_\delta(S)$. The promise means that all instances that are neither in S nor far from S (i.e., not in $\Gamma_\delta(S)$) are ignored, which is essential for sub-linear algorithms for natural problems. This makes the property testing task potentially easier than the corresponding standard decision task (cf. §10.1.2.2). To demonstrate the point, consider the set S consisting of strings that have a majority of 1's. Then, deciding membership in S requires linear time, because random n-bit long strings with $\lfloor n/2 \rfloor$ ones cannot be distinguished from random n-bit long strings with $\lfloor n/2 \rfloor + 1$ ones by probing a sub-linear number of locations (even if randomization and error probability are allowed – see Exercise 10.8). On the other hand, the fraction of 1's in the input can be approximated by a randomized polylogarithmic time algorithm (which yields a property tester for S; see Exercise 10.9). Thus, for some sets, deciding membership requires linear time, while property testing can be done in polylogarithmic time.

The essential role of randomization. Referring to the foregoing example, we note that randomization is essential for any sub-linear time algorithm that distinguishes this set S from, say, $\Gamma_{0.4}(S)$. Specifically, a sub-linear time deterministic algorithm cannot distinguish 1^n from any input that has 1's in each position probed by that algorithm on input 1^n. In general, on input x, a (sub-linear time) deterministic algorithm always reads the same bits of x and thus cannot distinguish x from any z that agrees with x on these bit locations.

Note that, in both cases, we are able to prove lower-bounds on the time complexity of algorithms. This success is due to the fact that these lower-bounds are actually information theoretic in nature; that is, these lower-bounds actually refer to the number of queries performed by these algorithms.

10.1.2.2 Two models for testing graph properties

In this subsection we consider the complexity of property testing for sets of graphs that are *closed under graph isomorphism*; such sets are called graph properties. In view of the importance of representation in the context of property testing, we consider two standard representations of graphs (cf. Appendix G.1), which indeed yield two different models of testing graph properties.

1. The adjacency matrix representation. Here a graph $G = ([N], E)$ is represented (in a somewhat redundant form) by an N-by-N Boolean matrix $M_G = (m_{i,j})_{i,j\in[N]}$ such that $m_{i,j} = 1$ if and only if $\{i,j\} \in E$.

2. Bounded incidence-lists representation. For a fixed parameter d, a graph $G = ([N], E)$ of degree at most d is represented (in a somewhat redundant form) by a mapping $\mu_G : [N] \times [d] \to [N] \cup \{\perp\}$ such that $\mu_G(u,i) = v$ if v is the i^{th} neighbor of u and $\mu_G(u,i) = \perp$ if v has less than i neighbors.

We stress that the aforementioned representations determine both the notion of distance between graphs and the type of queries performed by the algorithm. As we shall see, the difference between these two representations yields a big difference in the complexity of corresponding property testing problems.

Theorem 10.12 (property testing in the adjacency matrix representation): *For any fixed $\delta > 0$ and each of the following sets, there exists a polylogarithmic time randomized algorithm that solves the corresponding property testing problem* (with respect to δ).

- *For every fixed $k \geq 2$, the set of k-colorable graphs.*
- *For every fixed $\rho > 0$, the set of graphs having a clique* (resp., independent set) *of density ρ.*
- *For every fixed $\rho > 0$, the set of N-vertex graphs having a cut*[2] *with at least $\rho \cdot N^2$ edges.*

[2]A cut in a graph $G = ([N], E)$ is a partition $(S, [N] \setminus S)$ of the set of vertices and the edges of the cut are the edges with exactly one endpoint in S. A bisection is a cut of the graph to two parts of equal cardinality.

- *For every fixed $\rho > 0$, the set of N-vertex graphs having a bisection*[2] *with at most $\rho \cdot N^2$ edges.*

In contrast, for some $\delta > 0$, there exists a graph property in $\mathcal{NP}$ for which property testing (with respect to δ) *requires linear time.*

The testing algorithms use a constant number of queries, where this constant is polynomial in the constant $1/\delta$. We highlight the fact that exact decision procedure for the corresponding sets require a linear number of queries. The running time of the aforementioned algorithms hides a constant that is exponential in their query complexity (except for the case of 2-colorability where the hidden constant is polynomial in $1/\delta$). Note that such dependencies seem essential, since setting $\delta = 1/N^2$ regains the original (non-relaxed) decision problems (which, with the exception of 2-colorability, are all NP-complete). Turning to the lower-bound, we note that the graph property for which this bound is proved is not a natural one. Again, the lower-bound on the time complexity follows from a lower-bound on the query complexity.

Theorem 10.12 exhibits a dichotomy between graph properties for which property testing is possible by a constant number of queries and graph properties for which property testing requires a linear number of queries. A combinatorial characterization of the graph properties for which property testing is possible (in the adjacency matrix representation) when using a constant number of queries is known.[3] We note that the constant in this characterization may depend arbitrarily on δ (and indeed, in some cases, it is a function growing faster than a tower of $1/\delta$ exponents).

Turning back to Theorem 10.12, we note that the results regarding property testing for the sets corresponding to max-cut and min-bisection yield approximation algorithms with an additive error term (of δN^2). For dense graphs (i.e., N-vertex graphs having $\Omega(N^2)$ edges), this yields a constant factor approximation for the standard approximation problem (as in Definition 10.1). That is, for every constant $c > 1$, we obtain a *c-factor approximation* of the problem of maximizing the size of a cut (resp., minimizing the size of a bisection) *in dense graphs*. On the other hand, the result regarding clique yields a so called dual-approximation for maximum clique; that is, we approximate the minimum number of missing edges in the densest induced graph of a given size.

Indeed, Theorem 10.12 is meaningful only for dense graphs. The same holds, in general, for the adjacency matrix representation.[4] Also note that property testing is trivial, under the adjacency matrix representation, for any graph property S satisfying $\Gamma_{o(1)}(S) = \emptyset$ (e.g., the set of connected graphs, the set of Hamiltonian graphs, etc).

[3]Describing this fascinating result of Alon *et. al.* [8], which refers to the notion of regular partitions (introduced by Szemerédi), is beyond the scope of the current text.

[4]In this model, all N-vertex graphs having less than $(\delta/2) \cdot \binom{N}{2}$ edges may be accepted if and only if there exists such a (non-dense) graph in the predetermined set. This trivial decision regarding non-dense graphs is correct, because if the set S contains an N-vertex graph with less than $(\delta/2) \cdot \binom{N}{2}$ edges then $\Gamma_\delta(S)$ contains no N-vertex graph having less than $(\delta/2) \cdot \binom{N}{2}$ edges.

We now turn to the bounded incidence-lists representation, which is relevant only for bounded degree graphs. The problems of max-cut, min-bisection and clique (as in Theorem 10.12) are trivial under this representation, but graph connectivity becomes non-trivial, and the complexity of property testing for the set of bipartite graphs changes dramatically.

Theorem 10.13 (property testing in the bounded incidence-lists representation): *The following assertions refer to the representation of graphs by incidence-lists of length d.*

- *For any fixed d and $\delta > 0$, there exists a polylogarithmic time randomized algorithm that solves the property testing problem for the set of connected graphs of degree at most d.*
- *For any fixed d and $\delta > 0$, there exists a sub-linear randomized algorithm that solves the property testing problem for the set of bipartite graphs of degree at most d. Specifically, on input an N-vertex graph, the algorithm runs for $\widetilde{O}(\sqrt{N})$ time.*
- *For any fixed $d \geq 3$ and some $\delta > 0$, property testing for the set of N-vertex* (3-regular) *bipartite graphs requires $\Omega(\sqrt{N})$ queries.*
- *For some fixed d and $\delta > 0$, property testing for the set of N-vertex 3-colorable graphs requires $\Omega(N)$ queries.*

The running time of the algorithms hides a constant that is polynomial in $1/\delta$. Providing a characterization of graph properties according to the complexity of the corresponding tester (in the bounded incidence-lists representation) is an interesting open problem.

Decoupling the distance from the representation. So far, we have confined our attention to the Hamming distance between the representations of graphs. This made the choice of representation even more important than usual (i.e., more crucial than is common in complexity theory). In contrast, it is natural to consider a notion of distance between graphs that is independent of their representation. For example, the distance between $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ can be defined as the minimum of the size of symmetric difference between E_1 and the set of edges in a graph that is isomorphic to G_2. The corresponding relative distance may be defined as the distance divided by $|E_1| + |E_2|$ (or by $\max(|E_1|, |E_2|)$).

10.1.2.3 Beyond graph properties

Property testing has been applied to a variety of computational problems beyond the domain of graph theory. In fact, this area first emerged in the algebraic domain, where the instances (to be viewed as inputs to the testing algorithm) are functions and the relevant properties are sets of algebraic functions. The archetypical example is the set of low-degree polynomials; that is, m-variate polynomials of total (or individual) degree d over some finite field $\mathrm{GF}(q)$, where m, d and q are parameters that may depend on the length of the input (or satisfy some relationships; e.g., $q = d^3 = m^6$). Note that, in this case, the input is the description of a m-variate function over GF(q), which means that it has length $q^m \cdot \log_2 q$. Viewing the problem instance as a function suggests a natural measure of distance (i.e., the fraction of arguments on which the functions disagree) as well as a natural way of accessing the instance (i.e., querying the function for the value of selected arguments).

Note that we have referred to these computational problems, under a different terminology, in §9.3.2.2 and in §9.3.2.1. In particular, in §9.3.2.1 we refereed to the special case of linear Boolean functions (i.e., individual degree 1 and $q = 2$), whereas in §9.3.2.2 we used the setting $q = \text{poly}(d)$ and $m = d/\log d$ (where d is a bound on the total degree).

Other domains of computational problems in which property testing was studied include geometry (e.g., clustering problems), formal languages (e.g., testing membership in regular sets), coding theory (cf. Appendix E.1.2), probability theory (e.g., testing equality of distributions), and combinatorics (e.g., monotone and junta functions). As discuss at the end of §10.1.2.2, it is often natural to decouple the distance measure from the representation of the objects (i.e., the way of accessing the problem instance). This is done by introducing a representation-independent notion of distance between instances, which should be natural in the context of the problem at hand.

10.2 Average Case Complexity

Teaching note: We view average-case complexity as referring to the performance on average (or typical) instances, and not as the average performance on random instances. This choice is justified in §10.2.1.1. Thus, the current theory may be termed typical-case complexity. The term average-case is retained for historical reasons.

Our approach so far (including in Section 10.1) is termed worst-case complexity, because it refers to the performance of potential algorithms on each legitimate instance (and hence to the performance on the worst possible instance). That is, computational problems were defined as referring to a set of instances and performance guarantees were required to hold for each instance in this set. In contrast, average-case complexity allows ignoring a negligible measure of the possible instances, where *the identity of the ignored instances is determined by the analysis of potential solvers and not by the problem's statement.*

A few comments are in place. Firstly, as just hinted, the standard statement of the worst-case complexity of a computational problem (especially one having a promise) may also ignores some instances (i.e., those considered inadmissible or violating the promise), but these instances are determined by the problem's statement. In contrast, the inputs ignored in average-case complexity are not inadmissible in any inherent sense (and are certainly not identified as such by the problem's statement). It is just that they are viewed as exceptional when claiming that a specific algorithm solve the problem; furthermore, these exceptional instances are determined by the analysis of that algorithm. Needless to say, these exceptional instances ought to be rare (i.e., occur with negligible probability).

The last sentence raises a couple of issues. Firstly, a distribution on the set of admissible instances has to be specified. In fact, we shall consider a new type of computational problems, each consisting of a standard computational problem coupled with a probability distribution on instances. Consequently, the question of which distributions should be considered arises. This question and numerous other definitional issues will be addressed in §10.2.1.1.

Before proceeding, let us spell out the rather straightforward motivation to the study of the average-case complexity of computational problems. It is that, in real-life applications, one may be perfectly happy with an algorithm that solves the problem fast on almost all instances that arise in the application. That is, one may be willing to tolerate error provided that it occurs with negligible probability, where the probability is taken over the distribution of instances encountered in the application. We stress that a key aspect in this approach is a good modeling of the type of distributions of instances that are encountered in natural algorithmic applications.

At this point a natural question arises: *can natural computational problems be solve efficiently when restricting attention to typical instances?* The bottom-line of this section is that, for a well-motivated choice of definitions, our conjecture is that the "distributional version" of NP is not contained in the average-case (or typical-case) version of P. This means that some NP problems are not merely hard in the worst-case, but rather "typically hard" (i.e., hard on typical instances drawn from some simple distribution). Specifically, hard instances may occur in natural algorithmic applications (and not only in cryptographic (or other "adversarial") applications that are design on purpose to produce hard instances).[5] This conjecture motivates the development of an average-case analogue of NP-completeness, which will be presented in this section. Indeed, the entire section may be viewed as an average-case analogue of Chapter 2.

Organization. A major part of our exposition is devoted to the definitional issues that arise when developing a general theory of average-case complexity. These issues are discussed in §10.2.1.1. In §10.2.1.2 we prove the existence of a distributional problem that is "NP-complete" in the average-case complexity sense. In §10.2.1.3 we extend the treatment to randomized algorithms. Additional ramifications are presented in Section 10.2.2.

[5]We highlight two differences between the current context (of natural algorithmic applications) and the context of cryptography. Firstly, in the current context and when referring to problems that are typically hard, the simplicity of the underlying input distribution is of great concern: the simpler this distribution, the more appealing the hardness assertion becomes. This concern is irrelevant in the context of cryptography. On the other hand (see discussion at the beginning of Section 7.1.1 and/or at end of §10.2.2.2), cryptographic applications require the ability to efficiently generate hard instances *together with corresponding solutions*.

10.2.1 The basic theory

In this section we provide a basic treatment of the theory of average-case complexity, while postponing important ramifications to Section 10.2.2. The basic treatment consists of the preferred definitional choices for the main concepts as well as the identification of a complete problem for a natural class of average-case computational problems.

10.2.1.1 Definitional issues

The theory of average-case complexity is more subtle than may appear in first thought. In addition to the generic difficulty involved in defining relaxations, difficulties arise from the "interface" between standard probabilistic analysis and the conventions of complexity theory. This is most striking in the definition of the class of feasible average-case computations. Referring to the theory of worst-case complexity as a guideline, we shall address the following aspects of the analogous theory of average-case complexity.

1. *Setting the general framework.* We shall consider distributional problems, which are standard computational problems (see Section 1.2.2) coupled with distributions on the relevant instances.

2. *Identifying the class of feasible* (distributional) *problems.* Seeking an average-case analogue of classes such as $\mathcal{P}$, we shall reject the first definition that comes to mind (i.e., the naive notion of "average polynomial-time"), briefly discuss several related alternatives, and adopt one of them for the main treatment.

3. *Identifying the class of interesting* (distributional) *problems.* Seeking an average-case analogue of the class $\mathcal{NP}$, we shall avoid both the extreme of allowing arbitrary distributions (which collapses average-case complexity to worst-case complexity) and the opposite extreme of confining the treatment to a single distribution such as the uniform distribution.

4. *Developing an adequate notion of reduction among* (distributional) *problems.* As in the theory of worst-case complexity, this notion should preserve feasible solveability (in the current distributional context).

We now turn to the actual treatment of each of the aforementioned aspects.

Step 1: Defining distributional problems. Focusing on decision problems, we define distributional problems as pairs consisting of a decision problem and a probability ensemble.[6] For simplicity, here a probability ensemble $\{X_n\}_{n\in\mathbb{N}}$ is a

[6] We mention that even this choice is not evident. Specifically, Levin [145] (see discussion in [85]) advocates the use of a single probability distribution defined over the set of all strings. His argument is that this makes the theory less representation-dependent. At the time we were convinced of his argument (see [85]), but currently we feel that the representation-dependent effects discussed in [85] are legitimate. Furthermore, the alternative formulation of [85] comes across as unnatural and tends to confuse some readers.

sequence of random variables such that X_n ranges over $\{0,1\}^n$. Thus, $(S, \{X_n\}_{n\in\mathbb{N}})$ is the distributional problem consisting of the problem of deciding membership in the set S with respect to the probability ensemble $\{X_n\}_{n\in\mathbb{N}}$. (The treatment of search problem is similar; see §10.2.2.1.) We denote the uniform probability ensemble by $U = \{U_n\}_{n\in\mathbb{N}}$; that is, U_n is uniform over $\{0,1\}^n$.

Step 2: Identifying the class of feasible problems. The first idea that comes to mind is defining the problem $(S, \{X_n\}_{n\in\mathbb{N}})$ as feasible (on the average) if there exists an algorithm A that solves S such that the *average running time* of A on X_n is bounded by a polynomial in n (i.e., there exists a polynomial p such that $\mathsf{E}[t_A(X_n)] \leq p(n)$, where $t_A(x)$ denotes the running-time of A on input x). The problem with this definition is that it very sensitive to the model of computation and is not closed under algorithmic composition. Both deficiencies are a consequence of the fact that t_A may be polynomial on the average with respect to $\{X_n\}_{n\in\mathbb{N}}$ but t_A^2 may fail to be so (e.g., consider $t_A(x'x'') = 2^{|x'|}$ if $x' = x''$ and $t_A(x'x'') = |x'x''|^2$ otherwise, coupled with the uniform distribution over $\{0,1\}^n$). We conclude that the *average running-time* of algorithms is not a robust notion. We also doubt the naive appeal of this notion, and view the *typical running time* of algorithms (as defined next) as a more natural notion. Thus, we shall consider an algorithm as feasible if its running-time is typically polynomial.[7]

We say that A is typically polynomial-time on $X = \{X_n\}_{n\in\mathbb{N}}$ if there exists a polynomial p such that the probability that A runs more that $p(n)$ steps on X_n is *negligible* (i.e., for every polynomial q and all sufficiently large n it holds that $\mathsf{Pr}[t_A(X_n) > p(n)] < 1/q(n)$). The question is what is required in the "untypical" cases, and two possible definitions follow.

1. The simpler option is saying that $(S, \{X_n\}_{n\in\mathbb{N}})$ is (typically) feasible if there exists an algorithm A that solves S such that A is typically polynomial-time on $X = \{X_n\}_{n\in\mathbb{N}}$. This effectively requires A to *correctly solve S on each instance*, which is more than was required in the motivational discussion. (Indeed, if the underlying reasoning is ignoring rare cases, then we should ignore them altogether rather than ignoring them in a partial manner (i.e., only ignore their affect on the running-time).)

2. The alternative, which fits the motivational discussion, is saying that (S, X) is (typically) feasible if there exists an algorithm A such that A typically solves S on X in polynomial-time; that is, there exists a polynomial p such that *the probability that on input X_n algorithm A either errs or runs more that $p(n)$ steps is negligible*. This formulation totally ignores the untypical instances. Indeed, in this case we may assume, without loss of generality, that A always runs in polynomial-time (see Exercise 10.11), but we shall not

[7] An alternative choice, taken by Levin [145] (see discussion in [85]), is considering as feasible (w.r.t $X = \{X_n\}_{n\in\mathbb{N}}$) any algorithm that runs in time that is polynomial in a function that is linear on the average (w.r.t X); that is, requiring that there exists a polynomial p and a function $\ell : \{0,1\}^* \to \mathbb{N}$ such that $t(x) \leq p(\ell(x))$ and $\mathsf{E}[\ell(X_n)] = O(n)$. This definition is robust (i.e., it does not suffer from the aforementioned deficiencies) and is arguably as "natural" as the naive definition (i.e., $\mathsf{E}[t_A(X_n)] \leq \mathrm{poly}(n)$).

do so here (in order to facilitate viewing the first option as a special case of the current option).

We note that both alternatives actually define *typical* feasibility and not *average-case* feasibility. To illustrate the difference between the two options, consider the distributional problem of deciding whether a uniformly selected (n-vertex) graph contains a Hamiltonian path. Intuitively, this problem is "typically trivial" (with respect to the uniform distribution)[8] because the algorithm may always say `yes` and be wrong with exponentially vanishing probability. Indeed, this trivial algorithm is admissible by the second approach, but not by the first approach. In light of the foregoing, we adopt the second approach.

Definition 10.14 (the class tpc$\mathcal{P}$): *We say that A* typically solves $(S, \{X_n\}_{n\in\mathbb{N}})$ in polynomial-time *if there exists a polynomial p such that the probability that on input X_n algorithm A either errs or runs more that $p(n)$ steps is negligible.*[9] *We denote by* tpc$\mathcal{P}$ *the class of distributional problems that are typically solvable in polynomial-time.*

Clearly, for every $S \in \mathcal{P}$ and every probability ensemble X, it holds that $(S, X) \in$ tpc$\mathcal{P}$. However, tpc$\mathcal{P}$ contains also distributional problems (S, X) with $S \notin \mathcal{P}$ (see Exercises 10.12 and 10.13). The big question, which underlies the theory of average-case complexity, is whether *natural distributional versions* of $\mathcal{NP}$ are in tpc$\mathcal{P}$. Thus, we turn to identify such versions.

Step 3: Identifying the class of interesting problems. Seeking to identify reasonable distributional versions of $\mathcal{NP}$, we note that two extreme choices should be avoided. On one hand, we must limit the class of admissible distributions so to prevent the collapse of average-case complexity to worst-case complexity (by a selection of a pathological distribution that resides on the "worst case" instances). On the other hand, we should allow for various types of natural distributions rather than confining attention merely to the uniform distribution (which seems misguided by the naive belief by which this distribution is the only one relevant to applications). Recall that our aim is addressing all possible input distributions that may occur in applications, and thus there is no justification for confining attention to the uniform distribution. Still, arguably, the distributions occuring in applications are "relatively simple" and so we seek to identify a class of simple distributions. One such notion (of simple distributions) underlies the following definition, while a more liberal notion will be presented in §10.2.2.2.

Definition 10.15 (the class dist$\mathcal{NP}$): *We say that a probability ensemble $X = \{X_n\}_{n\in\mathbb{N}}$ is* simple *if there exists a polynomial time algorithm that, on any input*

[8] In contrast, testing whether a given graph contains a Hamiltonian path seems "typically hard" for other distributions (see Exercise 10.24). Needless to say, in the latter distributions both yes-instances and no-instances appear with noticeable probability.

[9] Recall that a function $\mu : \mathbb{N} \to \mathbb{N}$ is negligible if for every positive polynomial q and all sufficiently large n it holds that $\mu(n) < 1/q(n)$. We say that A errs on x if $A(x)$ differs from the indicator value of the predicate $x \in S$.

$x \in \{0,1\}^*$, *outputs* $\Pr[X_{|x|} \leq x]$, *where the inequality refers to the standard lexicographic order of strings. We denote by* dist$\mathcal{NP}$ *the class of distributional problems consisting of decision problems in* $\mathcal{NP}$ *coupled with simple probability ensembles.*

Note that the uniform probability ensemble is simple, but so are many other "simple" probability ensembles. Actually, it makes sense to relax the definition such that the algorithm is only required to output an approximation of $\Pr[X_{|x|} \leq x]$, say, to within a factor of $1 \pm 2^{-2|x|}$. We note that Definition 10.15 interprets simplicity in computational terms; specifically, as the feasibility of answering very basic questions regarding the probability distribution (i.e., determining the probability mass assigned to a single (n-bit long) string and even to an interval of such strings). This simplicity condition is closely related to being polynomial-time sampleable via a *monotone* mapping (see Exercise 10.14). In §10.2.2.2 we shall consider the more intuitive and robust class of all polynomial-time sampleable probability ensembles (and show that it contains all simple ensembles). We believe that the combination of the results presented in §10.2.1.2 and §10.2.2.2 retrospectively endorses the choice underlying Definition 10.15. We articulate this point next.

We note that enlarging the class of distributions weakens the *conjecture* that the corresponding class of distributional NP problems contains infeasible problems. On the other hand, the *conclusion* that a specific distributional problem is not feasible becomes stronger when the problem belongs to a smaller class that corresponds to a restricted definition of admissible distributions. The combined results of §10.2.1.2 and §10.2.2.2 assert that a conjecture that refers to the larger class of polynomial-time sampleable ensembles implies a conclusion that refers to a (very) simple probability ensemble (which resides in the smaller class). Thus, the current setting in which both the conjecture and the conclusion refer to simple probability ensembles may be viewed as just an intermediate step.

Indeed, the big question in the current context is whether dist$\mathcal{NP}$ is contained in tpc$\mathcal{P}$. A positive answer (especially if extended to sampleable ensembles) would deem the P-vs-NP Question of little practical significant. However, our daily experience as well as much research effort indicate that some NP problems are not merely hard in the worst-case, but rather "typically hard". This supports the *conjecture that* dist$\mathcal{NP}$ *is not contained in* tpc$\mathcal{P}$.

Needless to say, the latter conjecture implies $\mathcal{P} \neq \mathcal{NP}$, and thus we should not expect to see a proof of it. What we may hope to see is "dist$\mathcal{NP}$-complete" problems; that is, problems in dist$\mathcal{NP}$ that are not in tpc$\mathcal{P}$ unless the entire class dist$\mathcal{NP}$ is contained in tpc$\mathcal{P}$. An adequate notion of a reduction is used towards formulating this possibility (which in turn is captured by the notion of "dist$\mathcal{NP}$-complete" problems).

Step 4: Defining reductions among (distributional) problems. Intuitively, such reductions must preserve average-case feasibility. Thus, in addition to the standard conditions (i.e., that the reduction be efficiently computable and yield a correct result), we require that the reduction "respects" the probability distribution of the corresponding distributional problems. Specifically, the reduction should not map very likely instances of the first ("starting") problem to rare instances of the second ("target") problem. Otherwise, having a typically polynomial-time algorithm for the second distributional problem does not necessarily yield such an algorithm for the first distributional problem. Following is the adequate analogue of a Cook reduction (i.e., general polynomial-time reduction), where the analogue of a Karp-reduction (many-to-one reduction) can be easily derived as a special case.

Teaching note: One may prefer presenting in class only the special case of many-to-one reductions, which suffices for Theorem 10.17. See Footnote 11.

Definition 10.16 (reductions among distributional problems): *We say that the oracle machine M* reduces *the distributional problem (S, X) to the distributional problem (T, Y) if the following three conditions hold.*

1. Efficiency: *The machine M runs in polynomial-time.*[10]
2. Validity: *For every $x \in \{0,1\}^*$, it holds that $M^T(x) = 1$ if an only if $x \in S$, where $M^T(x)$ denotes the output of the oracle machine M on input x and access to an oracle for T.*
3. Domination:[11] *The probability that, on input X_n and oracle access to T, machine M makes the query y is upper-bounded by $\text{poly}(|y|) \cdot \mathsf{Pr}[Y_{|y|} = y]$. That is, there exists a polynomial p such that, for every $y \in \{0,1\}^*$ and every $n \in \mathbb{N}$, it holds that*

$$\mathsf{Pr}[Q(X_n) \ni y] \leq p(|y|) \cdot \mathsf{Pr}[Y_{|y|} = y], \tag{10.2}$$

where $Q(x)$ denotes the set of queries made by M on input x and oracle access to T.

In addition, we require that the reduction does not make too short queries; that is, there exists a polynomial p' such that if $y \in Q(x)$ then $p'(|y|) \geq |x|$.

The l.h.s. of Eq. (10.2) refers to the probability that, on input distributed as X_n, the reduction makes the query y. This probability is required not to exceed the probability that y occurs in the distribution $Y_{|y|}$ by more than a polynomial factor in $|y|$. In this case we say that the l.h.s. of Eq. (10.2) is dominated by $\mathsf{Pr}[Y_{|y|} = y]$.

Indeed, the domination condition is the only aspect of Definition 10.16 that extends beyond the worst-case treatment of reductions and refers to the distributional setting. The domination condition does not insist that the distribution induced by

[10] In fact, one may relax the requirement and only require that M is typically polynomial-time with respect to X. The validity condition may also be relaxed similarly.

[11] Let us spell out the meaning of Eq. (10.2) in the special case of many-to-one reductions (i.e., $M^T(x) = 1$ if and only if $f(x) \in T$, where f is a polynomial-time computable function): in this case $\mathsf{Pr}[Q(X_n) \ni y]$ is replaced by $\mathsf{Pr}[f(X_n) = y]$. Assuming that f is one-to-one, Eq. (10.2) simplifies to $\mathsf{Pr}[X_{|f^{-1}(y)|} = f^{-1}(y)] \leq p(|y|) \cdot \mathsf{Pr}[Y_{|y|} = y]$ for any y in the image of f. Indeed, nothing is required for y not in the image of f.

$Q(X)$ equals Y, but rather allows some slackness that, in turn, is bounded so to guarantee preservation of typical feasibility (see Exercise 10.15).[12]

We note that the reducibility arguments extensively used in Chapters 7 and 8 (see discussion in Section 7.1.2) are actually reductions in the spirit of Definition 10.16 (except that they refer to different types of computational tasks).

10.2.1.2 Complete problems

Recall that our conjecture is that dist$\mathcal{NP}$ is not contained in tpc$\mathcal{P}$, which in turn strengthens the conjecture $\mathcal{P} \neq \mathcal{NP}$ (making infeasibility a typical phenomenon rather than a worst-case one). Having no hope of proving that dist$\mathcal{NP}$ is not contained in tpc$\mathcal{P}$, we turn to the study of complete problems with respect to that conjecture. Specifically, we say that a distributional problem (S, X) is dist$\mathcal{NP}$-complete if $(S, X) \in$ dist$\mathcal{NP}$ and every $(S', X') \in$ dist$\mathcal{NP}$ is reducible to (S, X) (under Definition 10.16).

Recall that it is quite easy to prove the mere existence of NP-complete problems and many natural problems are NP-complete. In contrast, in the current context, establishing completeness results is quite hard. This should not be surprising in light of the restricted type of reductions allowed in the current context. The restriction (captured by the domination condition) requires that "typical" instances of one problem should not be mapped to "untypical" instances of the other problem. However, it is fair to say that standard Karp-reductions (used in establishing NP-completeness results) map "typical" instances of one problem to quite "bizarre" instances of the second problem. Thus, the current subsection may be viewed as a study of reductions that do not commit this sin.

Theorem 10.17 (dist$\mathcal{NP}$-completeness): *dist$\mathcal{NP}$ contains a distributional problem (T, Y) such that each distributional problem in* dist$\mathcal{NP}$ *is reducible* (per Definition 10.16) *to (T, Y). Furthermore, the reduction is deterministic and many-to-one.*

Proof: We start by introducing such a problem, which is a natural distributional version of the decision problem $S_{\mathtt{u}}$ (used in the proof of Theorem 2.18). Recall that $S_{\mathtt{u}}$ contains the instance $\langle M, x, 1^t\rangle$ if there exists $y \in \cup_{i\leq t}\{0,1\}^i$ such that M accepts the input pair (x, y) within t steps. We couple $S_{\mathtt{u}}$ with the "quasi-uniform" probability ensemble U' that assigns to the instance $\langle M, x, 1^t\rangle$ a probability mass proportional to $2^{-(|M|+|x|)}$. Specifically, for every $\langle M, x, 1^t\rangle$ it holds that

$$\Pr[U'_n = \langle M, x, 1^t\rangle] = \frac{2^{-(|M|+|x|)}}{\binom{n}{2}} \tag{10.3}$$

[12] We stress that the notion of domination is incomparable to the notion of statistical (resp., computational) indistinguishability. On one hand, domination is a local requirement (i.e., it compares the two distribution on a point-by-point basis), whereas indistinguishability is a global requirement (which allows rare exceptions). On the other hand, domination does not require approximately equal values, but rather a ratio that is bounded in one direction. Indeed, domination is not symmetric. We comment that a more relaxed notion of domination that allows rare violations (as in Footnote 10) suffices for the preservation of typical feasibility.

where $n \stackrel{\text{def}}{=} |\langle M, x, 1^t \rangle| \stackrel{\text{def}}{=} |M| + |x| + t$. Note that, under a suitable encoding, the ensemble U' is indeed simple.[13]

The reader can easily verify that the generic reduction used when reducing any set in $\mathcal{NP}$ to $S_{\mathtt{u}}$ (see the proof of Theorem 2.18), fails to reduce dist$\mathcal{NP}$ to $(S_{\mathtt{u}}, U')$. Specifically, in some cases (see next paragraph), these reductions do not satisfy the domination condition. Indeed, the difficulty is that we have to reduce all dist$\mathcal{NP}$ problems (i.e., pairs consisting of decision problems and simple distributions) to one single distributional problem (i.e., $(S_{\mathtt{u}}, U')$). Applying the aforementioned reductions, we end up with many distributional versions of $S_{\mathtt{u}}$, and furthermore the corresponding distributions are very different (and are not necessarily dominated by a single distribution).

Let us take a closer look at the aforementioned generic reduction, when applied to an arbitrary $(S, X) \in$ dist$\mathcal{NP}$. This reduction maps an instance x to a triple $(M_S, x, 1^{p_S(|x|)})$, where M_S is a machine verifying membership in S (while using adequate NP-witnesses) and p_S is an adequate polynomial. The problem is that x may have relatively large probability mass (i.e., it may be that $\mathsf{Pr}[X_{|x|} = x] \gg 2^{-|x|}$) while $(M_S, x, 1^{p_S(|x|)})$ has "uniform" probability mass (i.e., $\langle M_S, x, 1^{p_S(|x|)} \rangle$ has probability mass smaller than $2^{-|x|}$ in U'). This violates the domination condition (see Exercise 10.18), and thus an alternative reduction is required.

The key to the alternative reduction is an (efficiently computable) encoding of strings taken from an arbitrary *simple* distribution by strings that have a similar probability mass under the uniform distribution. This means that the encoding should shrink strings that have relatively large probability mass under the original distribution. Specifically, this encoding will map x (taken from the ensemble $\{X_n\}_{n \in \mathbb{N}}$) to a codeword x' of length that is upper-bounded by the logarithm of $1/\mathsf{Pr}[X_{|x|} = x]$, ensuring that $\mathsf{Pr}[X_{|x|} = x] = O(2^{-|x'|})$. Accordingly, the reduction will map x to a triple $(M_{S,X}, x', 1^{p'(|x|)})$, where $|x'| < O(1) + \log_2(1/\mathsf{Pr}[X_{|x|} = x])$ and $M_{S,X}$ is an algorithm that (given x' and x) first verifies that x' is a proper encoding of x and next applies the standard verification (i.e., M_S) of the problem S. Such a reduction will be shown to satisfy all three conditions (i.e., efficiency, validity, and domination). Thus, instead of forcing the structure of the original distribution X on the target distribution U', the reduction will incorporate the structure of X in the reduced instance. A key ingredient in making this possible is the fact that X is simple (as per Definition 10.15).

With the foregoing motivation in mind, we now turn to the actual proof; that is, proving that any $(S, X) \in$ dist$\mathcal{NP}$ is reducible to $(S_{\mathtt{u}}, U')$. The following technical lemma is the basis of the reduction. In this lemma as well as in the sequel, it will be convenient to consider the (accumulative) distribution function of the probability ensemble X. That is, we consider $\mu(x) \stackrel{\text{def}}{=} \mathsf{Pr}[X_{|x|} \le x]$, and note that $\mu : \{0,1\}^* \to [0,1]$ is polynomial-time computable (because X satisfies

[13] For example, we may encode $\langle M, x, 1^t \rangle$, where $M = \sigma_1 \cdots \sigma_k \in \{0,1\}^k$ and $x = \tau_1 \cdots \tau_\ell \in \{0,1\}^\ell$, by the string $\sigma_1\sigma_1 \cdots \sigma_k\sigma_k 01 \tau_1\tau_1 \cdots \tau_\ell\tau_\ell 01^t$. Then $\binom{n}{2} \cdot \mathsf{Pr}[U'_n \le \langle M, x, 1^t \rangle]$ equals $(i_{|M|,|x|,t} - 1) + 2^{-|M|} \cdot |\{M' \in \{0,1\}^{|M|} : M' < M\}| + 2^{-(|M|+|x|)} \cdot |\{x' \in \{0,1\}^{|x|} : x' \le x\}|$, where $i_{k,\ell,t}$ is the ranking of $\{k, k+\ell\}$ among all 2-subsets of $[k+\ell+t]$.

Definition 10.15).

Coding Lemma:[14] Let $\mu : \{0,1\}^* \to [0,1]$ be a polynomial-time computable function that is monotonically non-decreasing over $\{0,1\}^n$ for every n (i.e., $\mu(x') \leq \mu(x'')$ for any $x' < x'' \in \{0,1\}^{|x'|}$). For $x \in \{0,1\}^n \setminus \{0^n\}$, let $x-1$ denote the string preceding x in the lexicographic order of n-bit long strings. Then there exist an encoding function C_μ that satisfies the following three conditions.

1. Compression: For every x it holds that $|C_\mu(x)| \leq 1 + \min\{|x|, \log_2(1/\mu'(x))\}$, where $\mu'(x) \stackrel{\rm def}{=} \mu(x) - \mu(x-1)$ if $x \notin \{0\}^*$ and $\mu'(0^n) \stackrel{\rm def}{=} \mu(0^n)$ otherwise.
2. Efficient Encoding: The function C_μ is computable in polynomial-time.
3. Unique Decoding: For every $n \in \mathbb{N}$, when restricted to $\{0,1\}^n$, the function C_μ is one-to-one (i.e., if $C_\mu(x) = C_\mu(x')$ and $|x| = |x'|$ then $x = x'$).

Proof: The function C_μ is defined as follows. If $\mu'(x) \leq 2^{-|x|}$ then $C_\mu(x) = 0x$ (i.e., in this case x serves as its own encoding). Otherwise (i.e., $\mu'(x) > 2^{-|x|}$) then $C_\mu(x) = 1z$, where z is chosen such that $|z| \leq \log_2(1/\mu'(x))$ and the mapping of n-bit strings to their encoding is one-to-one. Loosely speaking, z is selected to equal the shortest binary expansion of a number in the interval $(\mu(x)-\mu'(x), \mu(x)]$. Bearing in mind that this interval has length $\mu'(x)$ and that the different intervals are disjoint, we obtain the desired encoding. Details follows.

We focus on the case that $\mu'(x) > 2^{-|x|}$, and detail the way that z is selected (for the encoding $C_\mu(x) = 1z$). If $x > 0^{|x|}$ and $\mu(x) < 1$, then we let z be the longest common prefix of the binary expansions of $\mu(x-1)$ and $\mu(x)$; for example, if $\mu(1010) = 0.10010$ and $\mu(1011) = 0.10101111$ then $C_\mu(1011) = 1z$ with $z = 10$. Thus, in this case $0.z1$ is in the interval $(\mu(x-1), \mu(x)]$ (i.e., $\mu(x-1) < 0.z1 \leq \mu(x)$). For $x = 0^{|x|}$, we let z be the longest common prefix of the binary expansions of 0 and $\mu(x)$ and again $0.z1$ is in the relevant interval (i.e., $(0, \mu(x)]$). Finally, for x such that $\mu(x) = 1$ and $\mu(x-1) < 1$, we let z be the longest common prefix of the binary expansions of $\mu(x-1)$ and $1-2^{-|x|-1}$, and again $0.z1$ is in $(\mu(x-1), \mu(x)]$ (because $\mu'(x) > 2^{-|x|}$ and $\mu(x-1) < \mu(x) = 1$ imply that $\mu(x-1) < 1 - 2^{-|x|} < \mu(x)$). Note that if $\mu(x) = \mu(x-1) = 1$ then $\mu'(x) = 0 < 2^{-|x|}$.

We now verify that the foregoing C_μ satisfies the conditions of the lemma. We start with the compression condition. Clearly, if $\mu'(x) \leq 2^{-|x|}$ then $|C_\mu(x)| = 1 + |x| \leq 1 + \log_2(1/\mu'(x))$. On the other hand, suppose that $\mu'(x) > 2^{-|x|}$ and let us focus on the sub-case that $x > 0^{|x|}$ and $\mu(x) < 1$. Let $z = z_1 \cdots z_\ell$ be the longest common prefix of the binary expansions of $\mu(x-1)$ and $\mu(x)$. Then, $\mu(x-1) = 0.z0u$ and $\mu(x) = 0.z1v$, where $u, v \in \{0,1\}^*$. We infer that

$$\mu'(x) \;=\; \mu(x) - \mu(x-1) \;\leq\; \left(\sum_{i=1}^{\ell} 2^{-i} z_i + \sum_{i=\ell+1}^{\text{poly}(|x|)} 2^{-i} \right) - \sum_{i=1}^{\ell} 2^{-i} z_i \;<\; 2^{-|z|},$$

[14] The lemma actually refers to $\{0,1\}^n$, for any fixed value of n, but the efficiency condition is stated more easily when allowing n to vary (and using the standard asymptotic analysis of algorithms). Actually, the lemma is somewhat easier to state and establish for polynomial-time computable functions that are monotonically non-decreasing over $\{0,1\}^*$ (rather than over $\{0,1\}^n$). See further discussion in Exercise 10.19.

and $|z| < \log_2(1/\mu'(x)) \leq |x|$ follows. Thus, $|C_\mu(x)| \leq 1 + \min(|x|, \log_2(1/\mu'(x)))$ holds in both cases. Clearly, C_μ can be computed in polynomial-time by computing $\mu(x-1)$ and $\mu(x)$. Finally, note that C_μ satisfies the unique decoding condition, by separately considering the two aforementioned cases (i.e., $C_\mu(x) = 0x$ and $C_\mu(x) = 1z$). Specifically, in the second case (i.e., $C_\mu(x) = 1z$), use the fact that $\mu(x-1) < 0.z1 \leq \mu(x)$. □

To obtain an encoding that is one-to-one when applied to strings of different lengths we augment C_μ in the obvious manner; that is, we consider $C'_\mu(x) \stackrel{\rm def}{=} (|x|, C_\mu(x))$, which may be implemented as $C'_\mu(x) = \sigma_1\sigma_1 \cdots \sigma_\ell\sigma_\ell 01 C_\mu(x)$ where $\sigma_1 \cdots \sigma_\ell$ is the binary expansion of $|x|$. Note that $|C'_\mu(x)| = O(\log|x|) + |C_\mu(x)|$ and that C'_μ is one-to-one.

The machine associated with (S, X). Let μ be the accumulative probability function associated with the probability ensemble X, and M_S be the polynomial-time machine that verifies membership in S while using adequate NP-witnesses (i.e., $x \in S$ if and only if there exists $y \in \{0,1\}^{\mathrm{poly}(|x|)}$ such that $M(x, y) = 1$). Using the encoding function C'_μ, we introduce an algorithm $M_{S,\mu}$ with the intension of reducing the distributional problem (S, X) to $(S_{\mathtt{u}}, U')$ such that all instances (of S) are mapped to triples in which the first element equals $M_{S,\mu}$. Machine $M_{S,\mu}$ is given an alleged encoding (under C'_μ) of an instance to S along with an alleged proof that the corresponding instance is in S, and verifies these claims in the obvious manner. That is, on input x' and $\langle x, y\rangle$, machine $M_{S,\mu}$ first verifies that $x' = C'_\mu(x)$, and next verifiers that $x \in S$ by running $M_S(x, y)$. Thus, $M_{S,\mu}$ verifies membership in the set $S' = \{C'_\mu(x) : x \in S\}$, while using proofs of the form $\langle x, y\rangle$ such that $M_S(x, y) = 1$ (for the instance $C'_\mu(x)$).[15]

The reduction. We maps an instance x (of S) to the triple $(M_{S,\mu}, C'_\mu(x), 1^{p(|x|)})$, where $p(n) \stackrel{\rm def}{=} p_S(n) + p_C(n)$ such that p_S is a polynomial representing the running-time of M_S and p_C is a polynomial representing the running-time of the encoding algorithm.

Analyzing the reduction. Our goal is proving that *the foregoing mapping constitutes a reduction of* (S, X) *to* $(S_{\mathtt{u}}, U')$. We verify the corresponding three requirements (of Definition 10.16).

1. Using the fact that C_μ is polynomial-time computable (and noting that p is a polynomial), it follows that the foregoing mapping can be computed in polynomial-time.
2. Recall that, on input $(x', \langle x, y\rangle)$, machine $M_{S,\mu}$ accepts if and only if $x' = C'_\mu(x)$ and M_S accepts (x, y) within $p_S(|x|)$ steps. Using the fact that $C'_\mu(x)$ uniquely determines x, it follows that $x \in S$ if and only if there exists a string y of length at most $p(|x|)$ such that $M_{S,\mu}$ accepts $(C'_\mu(x), \langle x, y\rangle)$ in at most

[15] Note that $|y| = \mathrm{poly}(|x|)$, but $|x| = \mathrm{poly}(|C'_\mu(x)|)$ does not necessarily hold (and so S' is not necessarily in $\mathcal{NP}$). As we shall see, the latter point is immaterial.

$p(|x|)$ steps. Thus, $x \in S$ if and only if $(M_{S,\mu}, C'_\mu(x), 1^{p(|x|)}) \in S_{\mathbf{u}}$, and the validity condition follows.

3. In order to verify the domination condition, we first note that the foregoing mapping is one-to-one (because the transformation $x \to C'_\mu(x)$ is one-to-one). Next, we note that it suffices to consider instances of $S_{\mathbf{u}}$ that have a preimage under the foregoing mapping (since instances with no preimage trivially satisfy the domination condition). Each of these instances (i.e., each image of this mapping) is a triple with the first element equal to $M_{S,\mu}$ and the second element being an encoding under C'_μ. By the definition of U', for every such image $\langle M_{S,\mu}, C'_\mu(x), 1^{p(|x|)}\rangle \in \{0,1\}^n$, it holds that

$$\begin{aligned}\mathsf{Pr}[U'_n = \langle M_{S,\mu}, C'_\mu(x), 1^{p(|x|)}\rangle] &= \binom{n}{2}^{-1} \cdot 2^{-(|M_{S,\mu}|+|C'_\mu(x)|)} \\ &> c \cdot n^{-2} \cdot 2^{-(|C_\mu(x)|+O(\log|x|))},\end{aligned}$$

where $c = 2^{-|M_{S,\mu}|-1}$ is a constant depending only on S and μ (i.e., on the distributional problem (S,X)). Thus, for some positive polynomial q, we have

$$\mathsf{Pr}[U'_n = \langle M_{S,\mu}, C'_\mu(x), 1^{p(|x|)}\rangle] > 2^{-|C_\mu(x)|}/q(n). \tag{10.4}$$

By virtue of the compression condition (of the Coding Lemma), we have $2^{-|C_\mu(x)|} \geq 2^{-1-\min(|x|,\log_2(1/\mu'(x)))}$. It follows that

$$2^{-|C_\mu(x)|} \geq \mathsf{Pr}[X_{|x|} = x]/2. \tag{10.5}$$

Recalling that x is the only preimage that is mapped to $\langle M_{S,\mu}, C'_\mu(x), 1^{p(|x|)}\rangle$ and combining Eq. (10.4) & (10.5), we establish the domination condition.

The theorem follows. ■

Reflections. The proof of Theorem 10.17 demonstrates the fact that the reduction used in the proof of Theorem 2.18 does not introduce much structure in the reduced instances (i.e., does not reduce the original problem to a "highly structured special case" of the target problem). Put in other words, unlike more advanced worst-case reductions, this reduction does not map "random" (i.e., uniformly distributed) instances to highly structured instances (which occur with negligible probability under the uniform distribution). Thus, the reduction used in the proof of Theorem 2.18 suffices for reducing any distributional problem in dist$\mathcal{NP}$ to a distributional problem consisting of $S_{\mathbf{u}}$ coupled with *some* simple probability ensemble (see Exercise 10.20).[16]

However, Theorem 10.17 states more than the latter assertion. That is, it states that any distributional problem in dist$\mathcal{NP}$ is reducible to the *same* distributional

[16] Note that this cannot be said of most known Karp-reductions, which do map random instances to highly structured ones. Furthermore, the same (structure creating property) holds for the reductions obtained by Exercise 2.19.

version of $S_{\mathtt{u}}$. Indeed, the effort involved in proving Theorem 10.17 was due to the need for mapping instances taken from any simple probability ensemble (which may not be the uniform ensemble) to instances distributed in a manner that is dominated by a single probability ensemble (i.e., the quasi-uniform ensemble U').

Once we have established the existence of one dist$\mathcal{NP}$-complete problem, we may establish the dist$\mathcal{NP}$-completeness of other problems (in dist$\mathcal{NP}$) by reducing some dist$\mathcal{NP}$-complete problem to them (and relying on the transitivity of reductions (see Exercise 10.17)). Thus, the difficulties encountered in the proof of Theorem 10.17 are no longer relevant. Unfortunately, a seemingly more severe difficulty arises: almost all know reductions in the theory of NP-completeness work by introducing much structure in the reduced instances (i.e., they actually reduce to highly structured special cases). Furthermore, this structure is too complex in the sense that the distribution of reduced instances does not seem simple (in the sense of Definition 10.15). Designing reductions that avoid the introduction of such structure has turned out to be quite difficult; still several such reductions are cited in [85].

10.2.1.3 Probabilistic versions

The definitions in §10.2.1.1 can be extended so that to account also for randomized computations. For example, extending Definition 10.14, we have:

Definition 10.18 (the class tpc$\mathcal{BPP}$): *For a probabilistic algorithm A, a Boolean function f, and a time-bound function $t:\mathbb{N}\to\mathbb{N}$, we say that the string x is t-*bad for A with respect to f *if with probability exceeding $1/3$, on input x, either $A(x) \neq f(x)$ or A runs more that $t(|x|)$ steps. We say that A* typically solves $(S, \{X_n\}_{n\in\mathbb{N}})$ in probabilistic polynomial-time *if there exists a polynomial p such that the probability that X_n is p-bad for A with respect to the characteristic function of S is negligible. We denote by* tpc$\mathcal{BPP}$ *the class of distributional problems that are typically solvable in probabilistic polynomial-time.*

The definition of reductions can be similarly extended. This means that in Definition 10.16, both $M^T(x)$ and $Q(x)$ (mentioned in Items 2 and 3, respectively) are random variables rather than fixed objects. Furthermore, validity is required to hold (for every input) only with probability 2/3, where the probability space refers only to the internal coin tosses of the reduction. Randomized reductions are closed under composition and preserve typical feasibility (see Exercise 10.21).

Randomized reductions allow the presentation of a dist$\mathcal{NP}$-complete problem that refers to the (perfectly) uniform ensemble. Recall that Theorem 10.17 establishes the dist$\mathcal{NP}$-completeness of $(S_{\mathtt{u}}, U')$, where U' is a quasi-uniform ensemble (i.e., $\mathsf{Pr}[U'_n = \langle M, x, 1^t\rangle] = 2^{-(|M|+|x|)}/\binom{n}{2}$, where $n = |\langle M, x, 1^t\rangle|$). We first note that $(S_{\mathtt{u}}, U')$ can be randomly reduced to $(S'_{\mathtt{u}}, U'')$, where $S'_{\mathtt{u}} = \{\langle M, x, z\rangle : \langle M, x, 1^{|z|}\rangle \in S_{\mathtt{u}}\}$ and $\mathsf{Pr}[U''_n = \langle M, x, z\rangle] = 2^{-(|M|+|x|+|z|)}/\binom{n}{2}$ for every $\langle M, x, z\rangle \in \{0,1\}^n$. The randomized reduction consists of mapping $\langle M, x, 1^t\rangle$ to $\langle M, x, z\rangle$, where z is uniformly selected in $\{0,1\}^t$. Recalling that $U = \{U_n\}_{n\in\mathbb{N}}$ denotes the uniform probability ensemble (i.e., U_n is uniformly distributed on strings of length n) and using a suitable encoding we get.

Proposition 10.19 *There exists* $S \in \mathcal{NP}$ *such that every* $(S', X') \in \text{dist}\mathcal{NP}$ *is randomly reducible to* (S, U).

Proof Sketch: By the forgoing discussion, every $(S', X') \in \text{dist}\mathcal{NP}$ is randomly reducible to $(S'_{\mathtt{u}}, U'')$, where the reduction goes through $(S_{\mathtt{u}}, U')$. Thus, we focus on reducing $(S'_{\mathtt{u}}, U'')$ to $(S''_{\mathtt{u}}, U)$, where $S''_{\mathtt{u}} \in \mathcal{NP}$ is defined as follows. The string $\text{bin}_\ell(|u|)\cdot\text{bin}_\ell(|v|)\cdot u\cdot v\cdot w$ is in $S''_{\mathtt{u}}$ if and only if $\langle u, v, w\rangle \in S'_{\mathtt{u}}$ and $\ell = \lceil \log_2 |uvw| \rceil + 1$, where $\text{bin}_\ell(i)$ denotes the ℓ-bit long binary encoding of the integer $i \in [2^{\ell-1}]$ (i.e., the encoding is padded with zeros to a total length of ℓ). The reduction maps $\langle M, x, z\rangle$ to the string $\text{bin}_\ell(|x|)\text{bin}_\ell(|M|)Mxz$, where $\ell = \lceil \log_2(|M| + |x| + |z|)\rceil + 1$. Noting that this reduction satisfies all conditions of Definition 10.16, the proposition follows. $\square$

10.2.2 Ramifications

In our opinion, the most problematic aspect of the theory described in Section 10.2.1 is the definition of simple probability ensembles, which in turn restricts the definition of dist$\mathcal{NP}$ (Definition 10.15). This restriction strengthens the conjecture that dist$\mathcal{NP}$ is not contained in tpc$\mathcal{BPP}$, which means that it weakens conditional results that are based on this conjecture. An appealing extension of the class dist$\mathcal{NP}$ is presented in §10.2.2.2, where it is shown that if the extended class is not contained in tpc$\mathcal{BPP}$ then dist$\mathcal{NP}$ itself is not contained in tpc$\mathcal{BPP}$. Thus, dist$\mathcal{NP}$-complete problems enjoy the benefit of both being in the more restricted class (i.e., dist$\mathcal{NP}$) and being hard as long as some problems in the extended class is hard.

Another extension appears in §10.2.2.1, where we extend the treatment from decision problems to search problems. This extension is motivated by the realization that search problem are actually of greater importance to real-life applications (cf. Section 2.1.1), and hence a theory motivated by real-life applications must address such problems, as we do next.

Prerequisites: For the technical development of §10.2.2.1, we assume familiarity with the notion of unique solution and results regarding it as presented in Section 6.2.3. For the technical development of §10.2.2.2, we assume familiarity with hashing functions as presented in Appendix D.2.

10.2.2.1 Search versus Decision

Indeed, as in the case of worst-case complexity, search problems are at least as important as decision problems. Thus, an average-case treatment of search problems is indeed called for. We first present distributional versions of $\mathcal{PF}$ and $\mathcal{PC}$ (cf. Section 2.1.1), following the underlying principles of the definitions of tpc$\mathcal{P}$ and dist$\mathcal{NP}$.

Definition 10.20 (the classes tpc$\mathcal{PF}$ and dist$\mathcal{PC}$): *As in Section 2.1.1, we consider only polynomially bounded search problems; that is, binary relations* $R \subseteq$

$\{0,1\}^* \times \{0,1\}^*$ *such that for some polynomial* q *it holds that* $(x,y) \in R$ *implies* $|y| \leq q(|x|)$. *Recall that* $R(x) \stackrel{\rm def}{=} \{y : (x,y) \in R\}$ *and* $S_R \stackrel{\rm def}{=} \{x : R(x) \neq \emptyset\}$.

- *A* distributional search problem *consists of a polynomially bounded search problem coupled with a probability ensemble.*
- *The class* tpc$\mathcal{PF}$ *consists of all distributional search problems that are typically solvable in polynomial-time. That is,* $(R, \{X_n\}_{n \in \mathbb{N}}) \in$ tpc$\mathcal{PF}$ *if there exists an algorithm* A *and a polynomial* p *such that the probability that on input* X_n *algorithm* A *either errs or runs more that* $p(n)$ *steps is negligible, where* A *errs on* $x \in S_R$ *if* $A(x) \notin R(x)$ *and errs on* $x \notin S_R$ *if* $A(x) \neq \perp$.
- *A distributional search problem* (R, X) *is in* dist$\mathcal{PC}$ *if* $R \in \mathcal{PC}$ *and* X *is simple* (as in Definition 10.15).

Likewise, the class tpc$\mathcal{BPPF}$ consists of all distributional search problems that are typically solvable in *probabilistic* polynomial-time (cf., Definition 10.18). The definitions of *reductions among distributional problems*, presented in the context of decision problem, extend to search problems.

Fortunately, as in the context of worst-case complexity, the study of distributional search problems "reduces" to the study of distributional decision problems.

Theorem 10.21 (reducing search to decision): dist$\mathcal{PC} \subseteq$ tpc$\mathcal{BPPF}$ *if and only if* dist$\mathcal{NP} \subseteq$ tpc$\mathcal{BPP}$. *Furthermore, every problem in* dist$\mathcal{NP}$ *is reducible to some problem in* dist$\mathcal{PC}$*, and every problem in* dist$\mathcal{PC}$ *is* randomly *reducible to some problem in* dist$\mathcal{NP}$.

Proof Sketch: The furthermore part is analogous to the actual contents of the proof of Theorem 2.6 (see also Step 1 in the proof of Theorem 2.15). Indeed the reduction of $\mathcal{NP}$ to $\mathcal{PC}$ presented in the proof of Theorem 2.6 extends to the current context. Specifically, for any $S \in \mathcal{NP}$, we consider a relation $R \in \mathcal{PC}$ such that $S = \{x : R(x) \neq \emptyset\}$, and note that, for any probability ensemble X, the identity transformation reduces (S, X) to (R, X).

A difficulty arises in the opposite direction. Recall that in the proof of Theorem 2.6 we reduced the search problem of $R \in \mathcal{PC}$ to deciding membership in $S'_R \stackrel{\rm def}{=} \{\langle x, y' \rangle : \exists y'' \text{ s.t. } (x, y'y'') \in R\} \in \mathcal{NP}$. The difficulty encountered here is that, on input x, this reduction makes queries of the form $\langle x, y' \rangle$, where y' is a prefix of some string in $R(x)$. These queries may induce a distribution that is not dominated by any simple distribution. Thus, we seek an alternative reduction.

As a warm-up, let us assume for a moment that R has unique solutions (in the sense of Definition 6.26); that is, for every x it holds that $|R(x)| \leq 1$. In this case we may easily reduce the search problem of $R \in \mathcal{PC}$ to deciding membership in $S''_R \in \mathcal{NP}$, *where* $\langle x, i, \sigma \rangle \in S''_R$ *if and only if* $R(x)$ *contains a string in which the* $i^{\rm th}$ *bit equals* σ. Specifically, on input x, the reduction issues the queries $\langle x, i, \sigma \rangle$, where $i \in [\ell]$ (with $\ell = \text{poly}(|x|)$) and $\sigma \in \{0,1\}$, which allows for determining the single string in the set $R(x) \subseteq \{0,1\}^\ell$ (whenever $|R(x)| = 1$). The point is that this reduction can be used to reduce any $(R, X) \in$ dist$\mathcal{PC}$ (having unique solutions) to

$(S''_R, X'') \in \mathrm{dist}\mathcal{NP}$, *where* X'' *equally distributes the probability mass of* x (under X) *to all the tuples* $\langle x, i, \sigma\rangle$*; that is, for every* $i \in [\ell]$ *and* $\sigma \in \{0,1\}$*, it holds that* $\Pr[X''_{|\langle x,i,\sigma\rangle|} = \langle x, i, \sigma\rangle]$ *equals* $\Pr[X_{|x|} = x]/2\ell$.

Unfortunately, in the general case, R may not have unique solutions. Nevertheless, applying the main idea that underlies the proof of Theorem 6.27, this difficulty can be overcome. We first note that the foregoing mapping of instances of the distributional problem $(R, X) \in \mathrm{dist}\mathcal{PC}$ to instances of $(S''_R, X'') \in \mathrm{dist}\mathcal{NP}$ satisfies the efficiency and domination conditions even in the case that R does not have unique solutions. What may possibly fail (in the general case) is the validity condition (i.e., if $|R(x)| > 1$ then we may fail to recover any element of $R(x)$).

Recall that the main part of the proof of Theorem 6.27 is a randomized reduction that maps instances of R to triples of the form (x, m, h) such that m is uniformly distributed in $[\ell]$ and h is uniformly distributed in a family of hashing function H_ℓ^m, where $\ell = \mathrm{poly}(|x|)$ and H_ℓ^m is as in Appendix D.2. Furthermore, if $R(x) \neq \emptyset$ then, with probability $\Omega(1/\ell)$ over the choices of $m \in [\ell]$ and $h \in H_\ell^m$, there exists a unique $y \in R(x)$ such that $h(y) = 0^m$. Defining $R'(x, m, h) \stackrel{\mathrm{def}}{=} \{y \in R : h(y) = 0^m\}$, this yields a randomized reduction of the search problem of R to the search problem of R' such that with noticeable probability[17] the reduction maps instances that have solutions to instances having a unique solution. Furthermore, this reduction can be used to reduce any $(R, X) \in \mathrm{dist}\mathcal{PC}$ to $(R', X') \in \mathrm{dist}\mathcal{PC}$, *where* X' *distributes the probability mass of* x (under X) *to all the triples* (x, m, h) *such that for every* $m \in [\ell]$ *and* $h \in H_\ell^m$ *it holds that* $\Pr[X'_{|(x,m,h)|} = (x, m, h)]$ *equals* $\Pr[X_{|x|} = x]/(\ell \cdot |H_\ell^m|)$. (Note that with a suitable encoding, X' is indeed simple.)

The theorem follows by combining the two aforementioned reductions. That is, we first apply the randomized reduction of (R, X) to (R', X'), and next reduce the resulting instance to an instance of the corresponding decision problem $(S''_{R'}, X'')$, where X'' is obtained by modifying X' (rather than X). The combined randomized mapping satisfies the efficiency and domination conditions, and is valid with noticeable probability. The error probability can be made negligible by straightforward amplification (see Exercise 10.21). $\square$

10.2.2.2 Simple versus sampleable distributions

Recall that the definition of simple probability ensembles (underlying Definition 10.15) requires that the accumulating distribution function is polynomial-time computable. Recall that $\mu : \{0,1\}^* \to [0,1]$ is called the accumulating distribution function of $X = \{X_n\}_{n\in\mathbb{N}}$ if for every $n \in \mathbb{N}$ and $x \in \{0,1\}^n$ it holds that $\mu(x) \stackrel{\mathrm{def}}{=} \Pr[X_n \leq x]$, where the inequality refers to the standard lexicographic order of n-bit strings.

As argued in §10.2.1.1, the requirement that the accumulating distribution function is polynomial-time computable imposes severe restrictions on the set of admissible ensembles. Furthermore, it seems that these simple ensembles are indeed

[17]Recall that the probability of an event is said to be noticeable (in a relevant parameter) if it is greater than the reciprocal of some positive polynomial. In the context of randomized reductions, the relevant parameter is the length of the input to the reduction.

"simple" in some intuitive sense and hence represent a minimalistic model of distributions that may occur in practice. Seeking a maximalistic model of distributions that occur in practice, we consider the notion of polynomial-time sampleable ensembles (underlying Definition 10.22). We believe that the class of such ensembles contains all distributions that may occur in practice, because we believe that the real world should be modeled as a *feasible* (rather than an arbitrary) randomized process

Definition 10.22 (sampleable ensembles and the class samp$\mathcal{NP}$): *We say that a probability ensemble* $X = \{X_n\}_{n\in\mathbb{N}}$ *is* (polynomial-time) sampleable *if there exists a probabilistic polynomial-time algorithm* A *such that for every* $x \in \{0,1\}^*$ *it holds that* $\Pr[A(1^{|x|}) = x] = \Pr[X_{|x|} = x]$. *We denote by* samp$\mathcal{NP}$ *the class of distributional problems consisting of decision problems in* $\mathcal{NP}$ *coupled with sampleable probability ensembles.*

We first note that all simple probability ensembles are indeed sampleable (see Exercise 10.22), and thus dist$\mathcal{NP} \subseteq$ samp$\mathcal{NP}$. On the other hand, it seems that there are sampleable probability ensembles that are not simple (see Exercise 10.23). In fact, extending the scope of distributional problems (from dist$\mathcal{NP}$ to samp$\mathcal{NP}$) allows proving that every NP-complete problem has a distributional version in samp$\mathcal{NP}$ that is dist$\mathcal{NP}$-hard (see Exercise 10.24). Furthermore, it is possible to prove that all natural NP-complete problem have distributional versions that are samp$\mathcal{NP}$-complete.

Theorem 10.23 (samp$\mathcal{NP}$-completeness): *Suppose that* $S \in \mathcal{NP}$ *and that every set in* $\mathcal{NP}$ *is reducible to* S *by a Karp-reduction that does not shrink the input. Then there exists a polynomial-time sampleable ensemble* X *such that any problem in* samp$\mathcal{NP}$ *is reducible to* (S, X)

The proof of Theorem 10.23 is based on the observation that *there exists a polynomial-time sampleable ensemble that dominates all polynomial-time sampleable ensembles.* The existence of this ensemble is based on the notion of a universal (sampling) machine. For further details see Exercise 10.25. (Recall that when proving Theorem 10.17, we did not establish an analogous result for simple ensembles (but rather capitalized on the universal nature of $S_{\mathtt{u}}$).)

Theorem 10.23 establishes a rich theory of samp$\mathcal{NP}$-completeness, but does not relate this theory to the previously presented theory of dist$\mathcal{NP}$-completeness (see Figure 10.1). This is done in the next theorem, which asserts that the existence of typically hard problems in samp$\mathcal{NP}$ implies their existence in dist$\mathcal{NP}$.

Theorem 10.24 (samp$\mathcal{NP}$-completeness versus dist$\mathcal{NP}$-completeness): *If* samp$\mathcal{NP}$ *is not contained in* tpc$\mathcal{BPP}$ *then* dist$\mathcal{NP}$ *is not contained in* tpc$\mathcal{BPP}$.

Thus, the two "typical-case complexity" versions of the P-vs-NP Question are equivalent. That is, if some "sampleable distribution" versions of NP are not typically feasible then some "simple distribution" versions of NP are not typically

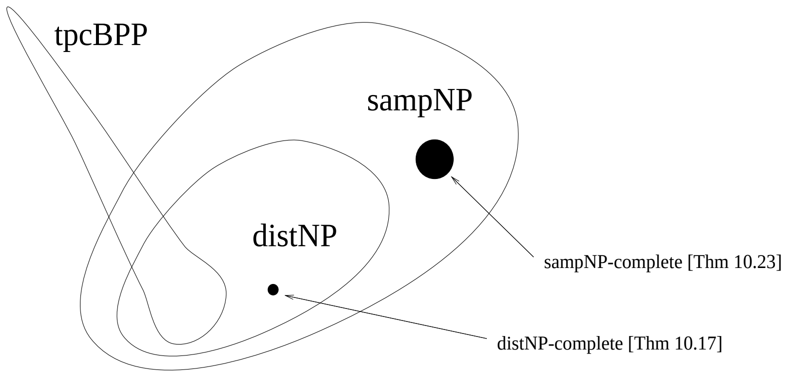


Figure 10.1: Two types of average-case completeness

feasible. In particular, if samp$\mathcal{NP}$-complete problems are not in tpc$\mathcal{BPP}$ then dist$\mathcal{NP}$-complete problems are not in tpc$\mathcal{BPP}$.

The foregoing assertions would all follow if samp$\mathcal{NP}$ were (randomly) reducible to dist$\mathcal{NP}$ (i.e., if every problem in samp$\mathcal{NP}$ were reducible (under a randomized version of Definition 10.16) to some problem in dist$\mathcal{NP}$); but, unfortunately, we do not know whether such reductions exist. Yet, underlying the proof of Theorem 10.24 is a more liberal notion of a reduction among distributional problem.

Proof Sketch: We shall prove that if dist$\mathcal{NP}$ is contained in tpc$\mathcal{BPP}$ then the same holds for samp$\mathcal{NP}$ (i.e., samp$\mathcal{NP}$ is contained in tpc$\mathcal{BPP}$). Actually, we shall show that if dist$\mathcal{PC}$ is contained in tpc$\mathcal{BPPF}$ then the sampleable version of dist$\mathcal{PC}$, denoted samp$\mathcal{PC}$, is contained in tpc$\mathcal{BPPF}$ (and refer to Exercise 10.26). Specifically, we shall show that under a relaxed notion of a randomized reduction, every problem in samp$\mathcal{PC}$ is reduced to some problem in dist$\mathcal{PC}$. Loosely speaking, this relaxed notion (of a randomized reduction) only requires that the validity and domination conditions (of Definition 10.16 (when adapted to randomized reductions)) hold with respect to a noticeable fraction of the probability space of the reduction.[18] We start by formulating this notion, when referring to distributional *search* problems.

Teaching note: The following proof is quite involved and is better left for advanced reading. Its main idea is related in one of the central ideas underlying the currently known proof of Theorem 8.11. This fact as well as numerous other applications of this idea, provide a good motivation for getting familiar with this idea.

[18] We warn that the existence of such a relaxed reduction between two specific distributional problems does not necessarily imply the existence of a corresponding (standard average-case) reduction. Specifically, although standard validity can be guaranteed (for problems in $\mathcal{PC}$) by repeated invocations of the reduction, such a process will *not* redeem the violation of the standard domination condition.

Definition: A relaxed reduction of the distributional problem (R, X) to the distributional problem (T, Y) is a probabilistic polynomial-time oracle machine M that satisfies the following conditions:

Notation: For every $x \in \{0,1\}^*$, we denote by $m(|x|) = \text{poly}(|x|)$ the number of internal coin tosses of M on input x, and denote by $M^T(x,r)$ the execution of M on input x, internal coins $r \in \{0,1\}^m$, and oracle access to T.

Validity: For some noticeable function $\rho : \mathbb{N} \to [0,1]$ (i.e., $\rho(n) > 1/\text{poly}(n)$) it holds that for every $x \in \{0,1\}^*$, there exists a set $\Omega_x \subseteq \{0,1\}^{m(|x|)}$ of size at least $\rho(|x|) \cdot 2^{m(|x|)}$ such that for every $r \in \Omega_x$ the reduction yields a correct answer (i.e., $M^T(x,r) \in R(x)$ if $R(x) \neq \emptyset$ and $M^T(x,r) = \perp$ otherwise).

Domination: There exists a positive polynomial p such that, for every $y \in \{0,1\}^*$ and every $n \in \mathbb{N}$, it holds that

$$\mathsf{Pr}[Q'(X_n) \ni y] \leq p(|y|) \cdot \mathsf{Pr}[Y_{|y|} = y], \tag{10.6}$$

where $Q'(x)$ is a random variable, defined over the set Ω_x (of the validity condition), representing the set of queries made by M on input x and oracle access to T. That is, $Q'(x)$ is defined by uniformly selecting $r \in \Omega_x$ and considering the set of queries made by M on input x, internal coins r, and oracle access to T. (In addition, as in Definition 10.16, we also require that the reduction does not make too short queries.)

The reader may verify that this relaxed notion of a reduction preserves typical feasibility; that is, for $R \in \mathcal{PC}$, if there exists a relaxed reduction of (R, X) to (T, Y) and (T, Y) is in tpc$\mathcal{BPPF}$ then (R, X) is in tpc$\mathcal{BPPF}$. The key observation is that the analysis may discard the case that, on input x, the reduction selects coins not in Ω_x. Indeed, the queries made in that case may be untypical and the answers received may be wrong, but this is immaterial. What matter is that, on input x, with noticeable probability the reduction selects coins in Ω_x, and produces "typical with respect to Y" queries (by virtue of the relaxed domination condition). Such typical queries are answered correctly by the algorithm that typically solves (T, Y), and if x has a solution then these answers yield a correct solution to x (by virtue of the relaxed validity condition). Thus, if x has a solution then with noticeable probability the reduction outputs a correct solution. On the other hand, the reduction never outputs a wrong solution (even when using coins not in Ω_x), because incorrect solutions are detected by relying on $R \in \mathcal{PC}$.

Our goal is presenting, for every $(R, X) \in \text{samp}\mathcal{PC}$, a relaxed reduction of (R, X) to a related problem $(R', X') \in \text{dist}\mathcal{PC}$, where (as usual) $X = \{X_n\}_{n\in\mathbb{N}}$ and $X' = \{X'_n\}_{n\in\mathbb{N}}$.

An oversimplified case: For starters, *suppose that X_n is uniformly distributed on some set $S_n \subseteq \{0,1\}^n$ and that there is a polynomial-time computable and invertible mapping μ of S_n to $\{0,1\}^{\ell(n)}$, where $\ell(n) = \log_2|S_n|$.* Then, mapping x to $1^{|x|-\ell(|x|)}0\mu(x)$, we obtain a reduction of (R, X) to (R', X'), where X'_{n+1} is uniform over $\{1^{n-\ell(n)}0v : v \in \{0,1\}^{\ell(n)}\}$ and $R'(1^{n-\ell(n)}0v) = R(\mu^{-1}(v))$ (or, equivalently,

$R(x) = R'(1^{|x|-\ell(|x|)}0\mu(x)))$. Note that X' is a simple ensemble and $R' \in \mathcal{PC}$; hence, $(R', X') \in \text{dist}\mathcal{PC}$. Also note that the foregoing mapping is indeed a valid reduction (i.e., it satisfies the efficiency, validity, and domination conditions). Thus, (R, X) is reduced to a problem in dist$\mathcal{PC}$ (and indeed the relaxation was not used here).

A simple but more instructive case: Next, we drop the assumption that there is a polynomial-time computable and invertible mapping μ of S_n to $\{0,1\}^{\ell(n)}$, but maintain the assumption that X_n is uniform on some set $S_n \subseteq \{0,1\}^n$ and assume that $|S_n| = 2^{\ell(n)}$ is easily computable (from n). In this case, we may map $x \in \{0,1\}^n$ to its image under a suitable randomly chosen hashing function h, which in particular maps n-bit strings to $\ell(n)$-bit strings. That is, we randomly map x to $(h, 1^{n-\ell(n)}0h(x))$, where h is uniformly selected in a set $H_n^{\ell(n)}$ of suitable hash functions (see Appendix D.2). This calls for redefining R' such that $R'(h, 1^{n-\ell(n)}0v)$ corresponds to the preimages of v under h that are in S_n. Assuming that h is a 1-1 mapping of S_n to $\{0,1\}^{\ell(n)}$, we may define $R'(h, 1^{n-\ell(n)}0v) = R(x)$ where x is the unique string satisfying $x \in S_n$ and $h(x) = v$, where the condition $x \in S_n$ may be *verified by providing the internal coins of the sampling procedure that generate* x. Denoting the sampling procedure of X by S, and letting $S(1^n, r)$ denote the output of S on input 1^n and internal coins r, we actually redefine R' as

$$R'(h, 1^{n-\ell(n)}0v) = \{\langle r, y\rangle : h(S(1^n, r)) = v \wedge y \in R(S(1^n, r))\}. \tag{10.7}$$

We note that $\langle r, y\rangle \in R'(h, 1^{|x|-\ell(|x|)}0h(x))$ yields a solution $y \in R(x)$ if $S(1^{|x|}, r) = x$, but otherwise "all bets are off" (as y will be a solution for $S(1^{|x|}, r) \neq x$). Now, although typically h will not be a 1-1 mapping of S_n to $\{0,1\}^{\ell(n)}$, it is the case that *for each $x \in S_n$, with constant probability over the choice of h, it holds that $h(x)$ has a unique preimage in S_n under h.* (See the proof of Theorem 6.27.) In this case $\langle r, y\rangle \in R'(h, 1^{|x|-\ell(|x|)}0h(x))$ implies $S(1^{|x|}, r) = x$ (which, in turn, implies $y \in R(x)$). We claim that *the randomized mapping of x to $(h, 1^{n-\ell(n)}0h(x))$, where h is uniformly selected in $H_{|x|}^{\ell(|x|)}$, yields a relaxed reduction of (R, X) to (R', X'), where $X'_{n'}$ is uniform over $H_n^{\ell(n)} \times \{1^{n-\ell(n)}0v : v \in \{0,1\}^{\ell(n)}\}$.* Needless to say, the claim refers to the reduction that makes the query $(h, 1^{n-\ell(n)}0h(x))$ and returns y if the oracle answer equals $\langle r, y\rangle$ and $y \in R(x)$.

The claim is proved by considering the set Ω_x of choices of $h \in H_{|x|}^{\ell(|x|)}$ for which $x \in S_n$ is the only preimage of $h(x)$ under h that resides in S_n (i.e., $|\{x' \in S_n : h(x') = h(x)\}| = 1$). In this case (i.e., $h \in \Omega_x$) it holds that $\langle r, y\rangle \in R'(h, 1^{|x|-\ell(|x|)}0h(x))$ implies that $S(1^{|x|}, r) = x$ and $y \in R(x)$, and the (relaxed) validity condition follows. The (relaxed) domination condition follows by noting that $\Pr[X_n = x] \approx 2^{-\ell(|x|)}$, that x is mapped to $(h, 1^{|x|-\ell(|x|)}0h(x))$ with probability $1/|H_{|x|}^{\ell(|x|)}|$, and that x is the only preimage of $(h, 1^{|x|-\ell(|x|)}0h(x))$ under the mapping (among $x' \in S_n$ such that $\Omega_{x'} \ni h$).

Before going any further, let us highlight the importance of hashing X_n to $\ell(n)$-bit strings. On one hand, this mapping is "sufficiently" one-to-one, and thus (with constant probability) the solution provided for the hashed instance (i.e., $h(x)$) yield a solution for the original instance (i.e., x). This guarantees the validity of the reduction. On the other hand, for a typical h, the mapping of X_n to $h(X_n)$ covers the relevant range almost uniformly. This guarantees that the reduction satisfies the domination condition. Note that these two phenomena impose conflicting requirements that are both met at the correct value of ℓ; that is, the one-to-one condition requires $\ell(n) \geq \log_2 |S_n|$, whereas an almost uniform cover requires $\ell(n) \leq \log_2 |S_n|$. Also note that $\ell(n) = \log_2(1/\mathsf{Pr}[X_n = x])$ for every x in the support of X_n; the latter quantity will be in our focus in the general case.

The general case: Finally, get rid of the assumption that X_n is *uniformly distributed* over some subset of $\{0,1\}^n$. All that we know is that there exists a probabilistic polynomial-time ("sampling") algorithm S such that $S(1^n)$ is distributed identically to X_n. In this (general) case, we map instances of (R,X) according to their probability mass such that x is mapped to an instance (of R') that consists of $(h,h(x))$ and additional information, where h is a random hash function mapping n-bit long strings to ℓ_x-bit long strings such that

$$\ell_x \stackrel{\text{def}}{=} \lceil \log_2(1/\mathsf{Pr}[X_{|x|} = x]) \rceil. \tag{10.8}$$

Since (in the general case) there may be more than 2^{ℓ_x} strings in the support of X_n, we need to augment the reduced instance in order to ensure that it is uniquely associated with x. The basic idea is augmenting the mapping of x to $(h,h(x))$ with additional information that restricts X_n to strings that occur with probability at least $2^{-\ell_x}$. Indeed, when X_n is restricted in this way, the value of $h(X_n)$ uniquely determines X_n.

Let $q(n)$ denote the randomness complexity of S and $S(1^n,r)$ denote the output of S on input 1^n and internal coin tosses $r \in \{0,1\}^{q(n)}$. Then, we randomly map x to $(h,h(x),h',v')$, where $h : \{0,1\}^{|x|} \to \{0,1\}^{\ell_x}$ and $h' : \{0,1\}^{q(|x|)} \to \{0,1\}^{q(|x|)-\ell_x}$ are random hash functions and $v' \in \{0,1\}^{q(|x|)-\ell_x}$ is uniformly distributed. The instance (h,v,h',v') of the redefined search problem R' has solutions that consists of pairs $\langle r,y \rangle$ such that $h(S(1^n,r)) = v \wedge h'(r) = v'$ and $y \in R(S(1^n,r))$. As we shall see, this augmentation guarantees that, with constant probability (over the choice of h,h',v'), the solutions to the reduced instance $(h,h(x),h',v')$ correspond to the solutions to the original instance x.

The foregoing description assumes that, on input x, we can determine ℓ_x, which is an assumption that cannot be justified. Instead, we select ℓ uniformly in $\{0,1,...,q(|x|)\}$, and so with noticeable probability we do select the correct value (i.e., $\mathsf{Pr}[\ell = \ell_x] = 1/(q(|x|)+1) = 1/\text{poly}(|x|)$). For clarity, we make n and ℓ explicit in the reduced instance. Thus, we randomly map $x \in \{0,1\}^n$ to $(1^n,1^\ell,h,h(x),h',v') \in \{0,1\}^{n'}$, where $\ell \in \{0,1,...,q(n)\}$, $h \in H_n^\ell$, $h' \in H_{q(n)}^{q(n)-\ell}$, and $v' \in \{0,1\}^{q(n)-\ell}$ are uniformly distributed in the corresponding sets.[19] This mapping will be used to reduce (R,X) to (R',X'), where R' and $X' = \{X'_{n'}\}_{n' \in \mathbb{N}}$

[19] As in other places, a suitable encoding will be used such that the reduction maps strings of the same length to strings of the same length (i.e., n-bit string are mapped to n'-bit strings, for $n' = \text{poly}(n)$). For example, we may encode $\langle 1^n,1^\ell,h,h(x),h',v' \rangle$ as $1^n 0 1^\ell 0 1^{q(n)-\ell} 0 \langle h \rangle \langle h(x) \rangle \langle h' \rangle \langle v' \rangle$, where each $\langle w \rangle$ denotes an encoding of w by a string of length $(n' - (n+q(n)+3))/4$.

are redefined (yet again). Specifically, we let

$$R'(1^n, 1^\ell, h, v, h', v') = \{\langle r, y\rangle : h(S(1^n, r)) = v \wedge h'(r) = v' \wedge y \in R(S(1^n, r))\} \quad (10.9)$$

and $X'_{n'}$ assigns equal probability to each $X_{n',\ell}$ (for $\ell \in \{0, 1, ..., n\}$), where each $X_{n',\ell}$ is isomorphic to the uniform distribution over $H_n^\ell \times \{0,1\}^\ell \times H_{q(n)}^{q(n)-\ell} \times \{0,1\}^{q(n)-\ell}$. Note that indeed $(R', X') \in \mathrm{dist}\mathcal{PC}$.

The aforementioned randomized mapping is analyzed by considering the correct choice for ℓ; that is, on input x, we focus on the choice $\ell = \ell_x$. Under this conditioning (as we shall show), *with constant probability over the choice of h, h' and v', the instance x is the only value in the support of X_n that is mapped to $(1^n, 1^{\ell_x}, h, h(x), h', v')$ and satisfies $\{r : h(S(1^n, r)) = h(x) \wedge h'(r) = v'\} \neq \emptyset$.* It follows that (for such h, h' and v') any solution $\langle r, y\rangle \in R'(1^n, 1^{\ell_x}, h, h(x), h', v')$ satisfies $S(1^n, r) = x$ and thus $y \in R(x)$, which means that the (relaxed) validity condition is satisfied. The (relaxed) domination condition is satisfied too, because (conditioned on $\ell = \ell_x$ and for such h, h', v') the probability that X_n is mapped to $(1^n, 1^{\ell_x}, h, h(x), h', v')$ approximately equals $\Pr[X'_{n',\ell_x} = (1^n, 1^{\ell_x}, h, h(x), h', v')]$.

We now turn to analyze the probability, over the choice of h, h' and v', that the instance x is the only value in the support of X_n that is mapped to $(1^n, 1^{\ell_x}, h, h(x), h', v')$ and satisfies $\{r : h(S(1^n, r)) = h(x) \wedge h'(r) = v'\} \neq \emptyset$. Firstly, we note that $|\{r : S(1^n, r) = x\}| \geq 2^{q(n)-\ell_x}$, and thus, with constant probability over the choice of $h' \in H_{q(n)}^{q(n)-\ell_x}$ and $v' \in \{0,1\}^{q(n)-\ell_x}$, there exists r that satisfies $S(1^n, r) = x$ and $h'(r) = v'$. Next, we note that, with constant probability over the choice of $h \in H_n^{\ell_x}$, it holds that x is the only string having probability mass at least $2^{-\ell_x}$ (under X_n) that is mapped to $h(x)$ under h. Finally, we prove that, with constant probability over the choice of $h \in H_n^{\ell_x}$ and $h' \in H_{q(n)}^{q(n)-\ell_x}$ (and even when conditioning on the previous items), the mapping $r \mapsto (h(S(1^n, r)), h'(r))$ maps the set $\{r : \Pr[X_n = S(1^n, r)] \leq 2^{-\ell_x}\}$ to $\{0,1\}^{q(n)}$ in an almost 1-1 manner. Specifically, with constant probability, no other r is mapped to the aforementioned pair $(h(x), v')$. Thus, the claim follows and so does the theorem. $\square$

Reflection. Theorem 10.24 implies that if samp$\mathcal{NP}$ is not contained in tpc$\mathcal{BPP}$ then every dist$\mathcal{NP}$-complete problem is not in tpc$\mathcal{BPP}$. This means that the hardness of some distributional problems that refer to sampleable distributions implies the hardness of some distributional problems that refer to simple distributions. Furthermore, by Proposition 10.19, this implies the hardness of distributional problems that refer to the uniform distribution. Thus, hardness with respect to some distribution in an utmost wide class (which arguably captures all distributions that may occur in practice) implies hardness with respect to a single simple distribution (which arguably is the simplest one).

Relation to one-way functions. We note that the existence of one-way functions (see Section 7.1) implies the existence of problems in samp$\mathcal{PC}$ that are not in tpc$\mathcal{BPPF}$ (which in turn implies the existence of such problems in dist$\mathcal{PC}$). Specifically, for a length-preserving one-way function f, consider the distributional search problem $(R_f, \{f(U_n)\}_{n\in\mathbb{N}})$, where $R_f = \{(f(r), r) : r \in \{0,1\}^*\}$.[20] On the other hand, it is not known whether the existence of a problem in samp$\mathcal{PC} \setminus$ tpc$\mathcal{BPPF}$ implies the existence of one-way functions. In particular, the existence of a problem (R, X) in samp$\mathcal{PC} \setminus$ tpc$\mathcal{BPPF}$ represents the feasibility of generating hard instances for the search problem R, whereas the existence of one-way function represents the feasibility of generating instance-solution pairs such that the instances are hard to solve (see Section 7.1.1). Indeed, the gap refers to whether or not *hard instances can be efficiently generated together with corresponding solutions.* Our world view is thus depicted in Figure 10.2, where lower levels indicate seemingly weaker assumptions.

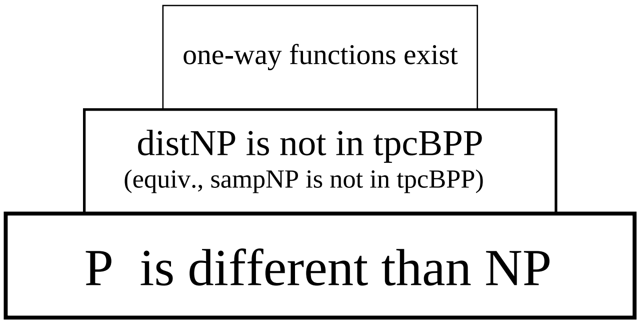


Figure 10.2: Worst-case vs average-case assumptions

Chapter Notes

In this chapter, we presented two different approaches to the relaxation of computational problems. The first approach refers to the concept of approximation, while the second approach refers to average-case analysis. We demonstrated that various natural notions of approximation can be cast within the standard frameworks, where the framework of promise problems (presented in Section 2.4.1) is the most non-standard framework we used (and it suffices for casting gap problems and property testing). In contrast, the study of average-case complexity requires the introduction of a new conceptual framework and addressing of various definitional issues.

A natural question at this point is what have we gained by relaxing the requirements. In the context of approximation, the answer is mixed: in some natural cases we gain a lot (i.e., we obtained feasible relaxations of hard problems), while in other natural cases we gain nothing (i.e., even extreme relaxations remain as intractable as the original version). In the context of average-case complexity, the negative side seems more prevailing (at least in the sense of being more systematic). In particular, assuming the existence of one-way functions, every natural

[20] Note that the distribution $f(U_n)$ is uniform in the special case that f is a permutation over $\{0,1\}^n$.

NP-complete problem has a distributional version that is hard, where this version refers to a sampleable ensemble. Furthermore, in this case, some problems in NP have hard distributional versions that refer to the uniform distribution.

Another difference between the two approaches is that the theory of approximation seems to lack a comprehensive structure, whereas the theory of average-case complexity seems to have a too rigid structure (which seems to foil attempts to present more appealing dist$\mathcal{NP}$-complete problems).

Approximation

The following bibliographic comments are quite laconic and neglect mentioning various important works (including credits for some of the results mentioned in our text). As usual, the interested reader is referred to corresponding surveys.

Search or Optimization. The interest in approximation algorithms increased considerably following the demonstration of the NP-completeness of many natural optimization problems. But, with some exceptions (most notably [167]), the systematic study of the complexity of such problems stalled till the discovery of the "PCP connection" (see Section 9.3.3) by Feige, Goldwasser, Lovász, and Safra [69]. Indeed the relatively "tight" inapproximation results for max-Clique, max-SAT, and the maximization of linear equations, due to Håstad [111, 112], build on previous work regarding PCP and their connection to approximation (cf., e.g., [70, 14, 13, 27, 173]). Specifically, Theorem 10.5 is due to [111], while Theorems 10.8 and 10.9 are due to [112]. The best known inapproximation result for minimum Vertex Cover (see Theorem 10.7) is due to [65], but we doubt it is tight (see, e.g., [134]). Reductions among approximation problems were defined and presented in [167]; see Exercise 10.7, which presents a major technique introduced in [167]. For general texts on approximation algorithms and problems (as discussed in Section 10.1.1), the interested reader is referred to the surveys collected in [117]. A compendium of NP optimization problems is available at [61].

Recall that a different type of approximation problems, which are naturally associated with search problems, were treated in Section 6.2.2. We note that an analogous definitional framework (e.g., gap problems, polynomial-time approximation schemes, etc) is applicable also to the approximate counting problems considered in Section 6.2.2.

Property testing. The study of property testing was initiated by Rubinfeld and Sudan [183] and re-initiated by Goldreich, Goldwasser, and Ron [93]. While the focus of [183] was on algebraic properties such as low-degree polynomials, the focus of [93] was on graph properties (and Theorem 10.12 is taken from [93]). The model of bounded-degree graphs was introduced in [99] and Theorem 10.13 combines results from [99, 100, 39]. For surveys of the area, the interested reader is referred to [73, 182].

Average-case complexity

The theory of average-case complexity was initiated by Levin [145], who in particular proved Theorem 10.17. In light of the laconic nature of the original text [145], we refer the interested reader to a survey [85], which provides a more detailed exposition of the definitions suggested by Levin as well as a discussion of the considerations underlying these suggestions. (This survey [85] provides also a brief account of further developments.)

As noted in §10.2.1.1, the current text uses a variant of the original definitions. In particular, our definition of "typical-case feasibility" differs from the original definition of "average-case feasibility" in totally discarding exceptional instances and in even allowing the algorithm to fail on them (and not merely run for an excessive amount of time). The alternative definition was suggested by several researchers, and appears as a special case of the general treatment provided in [41].

Section 10.2.2 is based on [28, 120]. Specifically, Theorem 10.21 (or rather the reduction of search to decision) is due to [28] and so is the introduction of the class samp$\mathcal{NP}$. A version of Theorem 10.24 was proven in [120], and our proof follows their ideas, which in turn are closely related to the ideas underlying the proof of Theorem 8.11 (proved in [113]).

Recall that we know of the existence of problems in dist$\mathcal{NP}$ that are hard provided samp$\mathcal{NP}$ contains hard problems. However, these problems refer to somewhat generic decision problems such as $S_{\mathtt{u}}$. The presentation of dist$\mathcal{NP}$-complete problems that combine a more natural decision problem (like `SAT` or `Clique`) with a simple probability ensemble is an open problem.

Exercises

Exercise 10.1 (general TSP) For any function g, prove that the following approximation problem is NP-Hard. Given a general TSP instance I, represented by a symmetric matrix of pairwise distances, the task is finding a tour of length that is at most a factor $g(I)$ of the minimum. Show that the result holds with $g(I) = \exp(\text{poly}(|I|))$ and for instances in which all distances are positive,

Guideline: By reduction from Hamiltonian path. Specifically, reduce the instance $G = ([n], E)$ to an n-by-n distance matrix $D = (d_{i,j})_{i,j\in[n]}$ such that $d_{i,j} = \exp(\text{poly}(n))$ if $\{i,j\} \in E$ and $d_{i,j} = 1$.

Exercise 10.2 (TSP with triangle inequalities) Provide a polynomial-time 2-factor approximation for the special case of TSP in which the distances satisfy the triangle inequality.

Guideline: First note that the length of any tour is lower-bounded by the weight of a minimum spanning tree in the corresponding weighted graph. Next note that such a tree yields a tour (of length twice the weight of this tree) that may visit some points several times. The triangle inequality guarantees that the tour does not become longer by "shortcuts" that eliminate multiple visits at the same point.

Exercise 10.3 (a weak version of Theorem 10.5) Using Theorem 9.16 prove that, for some constants $0 < a < b < 1$ when setting $L(N) = N^b$ and $s(N) = N^a$, it holds that $\texttt{gapClique}_{L,s}$ is NP-hard.

Guideline: Starting with Theorem 9.16, apply the Expander Random Walk Generator (of Proposition 8.29) in order to derive a PCP system with logarithmic randomness and query complexities that accepts no-instances of length n with probability at most $1/n$. The claim follows by applying the FGLSS-reduction (of Exercise 9.14), while noting that x is reduced to a graph of size poly$(|x|)$ such that the gap between yes and no-instances is at least a factor of $|x|$.

Exercise 10.4 (a weak version of Theorem 10.7) Using Theorem 9.16 prove that, for some constants $0 < s < L < 1$, the problem $\texttt{gapVC}_{s,L}$ is NP-hard.

Guideline: Note that combining Theorem 9.16 and Exercise 9.14 implies that for some constants $b < 1$ it holds that $\texttt{gapClique}_{L,s}$ is NP-hard, where $L(N) = b \cdot N$ and $s(N) = (b/2) \cdot N$. The claim follows using the relations between cliques, independent sets, and vertex covers.

Exercise 10.5 (a weak version of Theorem 10.9) Using Theorem 9.16 prove that, for some constants $0.5 < s < L < 1$, the problem $\texttt{gapLin}_{L,s}$ is NP-hard.

Guideline: Recall that by Theorems 9.16 and 9.21, the gap problem $\texttt{gapSAT}_\varepsilon^3$ is NP-Hard. Note that the result holds even if we restrict the instances to have exactly three (not necessarily different) literals in each clause. Applying the reduction of Exercise 2.26, note that, for any assignment τ, a clause that is satisfied by τ is mapped to seven equations of which exactly three are violated by τ, whereas a clause that is not satisfied by τ is mapped to seven equations that are all violated by τ.

Exercise 10.6 (natural inapproximability without the PCP Theorem) In contrast to the inapproximability results reviewed in §10.1.1.2, the NP-completeness of the following gap problem can be established (rather easily) without referring to the PCP Theorem. The instances of this problem are systems of quadratic equations over GF(2) (as in Exercise 2.27), yes-instances are systems that have a solution, and no-instances are systems for which any assignment violates at least one third of the equations.

Guideline: By Exercise 2.27, when given such a quadratic system, it is NP-hard to determine whether or not there exists an assignment that satisfies all the equations. Using an adequate small-bias generator (cf. Section 8.6.2), present an amplifying reduction (cf. Section 9.3.3) of the foregoing problem to itself. Specifically, if the input system has m equations then we use a generator that defines a sample space of poly(m) many m-bit strings, and consider the corresponding linear combinations of the input equations. Note that it suffices to bound the bias of the generator by $1/6$, whereas using an ε-biased generator yields an analogous result with $1/3$ replaced by $0.5 - \varepsilon$.

Exercise 10.7 (enforcing multi-way equalities via expanders) The aim of this exercise is presenting a major technique of Papadimitriou and Yannakakis [167], which is useful for designing reductions among approximation problems. Recalling that $\texttt{gapSAT}^3_{0.1}$ is NP-hard, our goal is proving NP-hard of the following gap problem, denoted $\texttt{gapSAT}^{3,c}_{\varepsilon}$, which is a special case of $\texttt{gapSAT}^3_{\varepsilon}$. Specifically, the instances are restricted to 3CNF formulae with each variable appearing in at most c clauses, where c (as ε) is a fixed constant. Note that the standard reduction of 3SAT to the corresponding special case (see proof of Proposition 2.22) does not preserve an approximation gap.[21] The idea is enforcing equality of the values assigned to the auxiliary variables (i.e., the copies of each original variable) by introducing equality constraints only for pairs of variables that correspond to edges of an expander graph (see Appendix E.2). For example, we enforce equality among the values of $z^{(1)},...,z^{(m)}$ by adding the clauses $z^{(i)} \vee \neg z^{(j)}$ for every $\{i,j\} \in E$, where E is the set of edges of am m-vertex expander graph. Prove that, for some constants c and $\varepsilon > 0$, the corresponding mapping reduces $\texttt{gapSAT}^3_{0.1}$ to $\texttt{gapSAT}^{3,c}_{\varepsilon}$.

Guideline: Using d-regular expanders, we map 3CNF to instances in which each variable appears in at most $2d+1$ clauses. Note that the number of added clauses is linearly related to the number of original clauses. Clearly, if the original formula is satisfiable then so is the reduced one. On the other hand, consider an arbitrary assignment τ' to the reduced formula ϕ' (i.e., the formula obtained by mapping ϕ). For each original variable z, if τ' assigns the same value to almost all copies of z then we consider the corresponding assignment in ϕ. Otherwise, by virtue of the added clauses, τ' does not satisfy a constant fraction of the clauses containing a copy of z.

Exercise 10.8 (deciding majority requires linear time) Prove that deciding majority requires linear-time even in a direct access model and when using a randomized algorithm that may err with probability at most 1/3.

Guideline: Consider the problem of distinguishing X_n from Y_n, where X_n (resp., Y_n) is uniformly distributed over the set of n-bit strings having exactly $\lfloor n/2 \rfloor$ (resp., $\lfloor n/2 \rfloor + 1$) ones. For any fixed set $I \subset [n]$, denote the projection of X_n (resp., Y_n) on I by X'_n (resp., Y'_n). Prove that the statistical difference between X'_n and Y'_n is bounded by $O(|I|/n)$. Note that the argument needs to be extended to the case that the examined locations are selected adaptively.

Exercise 10.9 (testing majority in polylogarithmic time) Show that testing majority (with respect to δ) can be done in polylogarithmic time by probing the input at a constant number of randomly selected locations.

---

[21] Recall that in this reduction each occurrence of each Boolean variable is replaced by a new copy of this variable, and clauses are added for enforcing the assignment of the same value to all these copies. Specifically, the m occurrence of variable z are replaced by the variables $z^{(1)},...,z^{(m)}$, while adding the clauses $z^{(i)} \vee \neg z^{(i+1)}$ and $z^{(i+1)} \vee \neg z^{(i)}$ (for $i = 1,...,m-1$). The problem is that almost all clauses of the reduced formula may be satisfied by an assignment in which half of the copies of each variable are assigned one value and the rest are assigned an opposite value. That is, an assignment in which $z^{(1)} = \cdots = z^{(i)} \neq z^{(i+1)} = \cdots = z^{(m)}$ violates only one of the auxiliary clauses introduced for enforcing equality among the copies of z. Using an alternative reduction that adds the clauses $z^{(i)} \vee \neg z^{(j)}$ for every $i,j \in [m]$ will not do either, because the number of added clauses may be quadratic in the number of original clauses.

Exercise 10.10 (testing Eulerian graphs in the adjacency matrix representation) Show that in this model the set of Eulerian graphs can be tested in polylogarithmic time.

Guideline: Focus on testing the set of graphs in which each vertex has an even degree. Note that, *in general*, the fact that the sets S' and S'' are testable within some complexity does *not* imply the same for the set $S' \cap S''$.

Exercise 10.11 (an equivalent definition of $\text{tpc}\mathcal{P}$**)** Prove that $(S, X) \in \text{tpc}\mathcal{P}$ if and only if there exists a polynomial-time algorithm A such that the probability that $A(X_n)$ errs (in determining membership in S) is a negligible function in n.

Exercise 10.12 ($\text{tpc}\mathcal{P}$ **versus** $\mathcal{P}$ **– Part 1)** Prove that $\text{tpc}\mathcal{P}$ contains a problem (S, X) such that S is not even recursive. Furthermore, use $X = U$.

Guideline: Let $S = \{0^{|x|}x : x \in S'\}$, where S' is an arbitrary (non-recursive) set.

Exercise 10.13 ($\text{tpc}\mathcal{P}$ **versus** $\mathcal{P}$ **– Part 2)** Prove that there exists a distributional problem (S, X) such that $S \notin \mathcal{P}$ and yet there exists an algorithm solving S (correctly on all inputs) in time that is typically polynomial with respect to X. Furthermore, use $X = U$.

Guideline: For any time-constructible function $t : \mathbb{N} \to \mathbb{N}$ that is super-polynomial and sub-exponential, use $S = \{0^{|x|}x : x \in S'\}$ for any $S' \in \text{DTIME}(t) \setminus \mathcal{P}$.

Exercise 10.14 (simple distributions and monotone sampling) We say that a probability ensemble $X = \{X_n\}_{n \in \mathbb{N}}$ is polynomial-time sampleable via a monotone mapping if there exists a polynomial p and a polynomial-time computable function f such that the following two conditions hold:

1. For every n, the random variables $f(U_{p(n)})$ and X_n are identically distributed.
2. For every n and every $r' < r'' \in \{0,1\}^{p(n)}$ it holds that $f(r') \leq f(r'')$, where the inequalities refers to the standard lexicographic order of strings.

Prove that X is simple if and only if it is polynomial-time sampleable via a monotone mapping.

Guideline: Suppose that X is simple, and let p be a polynomial bounding the running-time of the algorithm that on input x outputs $\Pr[X_{|x|} \leq x]$. Consider a mapping, denoted π, of $[0,1]$ to $\{0,1\}^n$ such that $r \in [0,1]$ is mapped to $x \in \{0,1\}^n$ if and only if $r \in [\Pr[X_n < x], \Pr[X_n \leq x])$. The desired function $f : \{0,1\}^{p(n)} \to \{0,1\}^n$ can be obtained from π by considering the binary representation of the numbers in $[0,1]$ (and recalling that the binary representation of $\Pr[X_{|x|} \leq x]$ has length at most $p(|x|)$). Note that f can be computed by binary search, using the fact that X is simple. Turning to the opposite direction, we note that any efficiently computable and monotone mapping $f : \{0,1\}^{p(n)} \to \{0,1\}^n$ can be efficiently inverted by a binary search. Furthermore, similar methods allow for efficiently determining the interval of $p(n)$-bit long strings that are mapped to any given n-bit long string.

Exercise 10.15 (reductions preserve typical polynomial-time solveability) Prove that if the distributional problem (S,X) is reducible to the distributional problem (S',X') and $(S',X') \in \text{tpc}\mathcal{P}$, then (S,X) is in $\text{tpc}\mathcal{P}$.

Guideline: Let B' denote the set of exceptional instances for the distributional problem (S',X'); that is, B' is the set of instances on which the solver in the hypothesis either errs or exceeds the typical running-time. Prove that $\mathsf{Pr}[Q(X_n) \cap B' \neq \emptyset]$ is a negligible function (in n), using both $\mathsf{Pr}[y \in Q(X_n)] \leq p(|y|) \cdot \mathsf{Pr}[X'_{|y|} = y]$ and $|x| \leq p'(|y|)$ for every $y \in Q(x)$. Specifically, use the latter condition for inferring that $\sum_{y \in B'} \mathsf{Pr}[y \in Q(X_n)]$ equals $\sum_{y \in \{y' \in B' : p'(|y'|) \geq n\}} \mathsf{Pr}[y \in Q(X_n)]$, which guarantees that a negligible function in $|y|$ for any $y \in Q(X_n)$ is negligible in n.

Exercise 10.16 (reductions preserve error-less solveability) In continuation to Exercise 10.15, prove that reductions preserve error-less solveability (i.e., solveability by algorithms that never err and typically run in polynomial-time).

Exercise 10.17 (transitivity of reductions) Prove that reductions among distributional problems (as in Definition 10.16) are transitive.

Guideline: The point is establishing the domination property of the composed reduction. The hypothesis that reductions do not make too short queries is instrumental here.

Exercise 10.18 For any $S \in \mathcal{NP}$ present a simple probability ensemble X such that the generic reduction used in the proof of Theorem 2.18, when applied to (S,X), violates the domination condition with respect to $(S_{\mathtt{u}}, U')$.

Guideline: Consider $X = \{X_n\}_{n\in\mathbb{N}}$ such that X_n is uniform over $\{0^{n/2}x' : x' \in \{0,1\}^{n/2}\}$.

Exercise 10.19 (variants of the Coding Lemma) Prove the following two variants of the Coding Lemma (which is stated in the proof of Theorem 10.17).

1. A variant that refers to any efficiently computable function $\mu : \{0,1\}^* \to [0,1]$ that is monotonically non-decreasing over $\{0,1\}^*$ (i.e., $\mu(x') \leq \mu(x'')$ for any $x' < x'' \in \{0,1\}^*$). That is, unlike in the proof of Theorem 10.17, here it holds that $\mu(0^{n+1}) \geq \mu(1^n)$ for every n.
2. As in Part 1, except that in this variant the function μ is strictly increasing and the compression condition requires that $|C_\mu(x)| \leq \log_2(1/\mu'(x))$ rather than $|C_\mu(x)| \leq 1 + \min\{|x|, \log_2(1/\mu'(x))\}$, where $\mu'(x) \stackrel{\text{def}}{=} \mu(x) - \mu(x-1)$.

In both cases, the proof is less cumbersome than the one presented in the main text.

Exercise 10.20 Prove that for any problem (S,X) in $\text{dist}\mathcal{NP}$ there exists a simple probability ensemble Y such that the reduction used in the proof of Theorem 2.18 suffices for reducing (S,X) to $(S_{\mathtt{u}}, Y)$.

Guideline: Consider $Y = \{Y_n\}_{n\in\mathbb{N}}$ such that Y_n assigns to the instance $\langle M, x, 1^t\rangle$ a probability mass proportional to $\pi_x \stackrel{\text{def}}{=} \mathsf{Pr}[X_{|x|} = x]$. Specifically, for every $\langle M, x, 1^t\rangle$ it holds that $\Pr[Y_n = \langle M, x, 1^t \rangle] = 2^{-|M|} \cdot \pi_x / \binom{n}{2}$, where $n \stackrel{\rm def}{=} |\langle M, x, 1^t \rangle| \stackrel{\rm def}{=} |M| + |x| + t$. Alternatively, we may set $\Pr[Y_n = \langle M, x, 1^t \rangle] = \pi_x$ if $M = M_S$ and $t = p_S(|x|)$ and $\Pr[Y_n = \langle M, x, 1^t \rangle] = 0$ otherwise, where M_S and P_S are as in the proof of Theorem 2.18.

Exercise 10.21 (randomized reductions) Following the outline in §10.2.1.3, provide a definition of randomized reductions among distributional problems.

1. In analogy to Exercise 10.15, prove that randomized reductions preserve feasible solveability (i.e., typical solveability in probabilistic polynomial-time). That is, if the distributional problem (S, X) is randomly reducible to the distributional problem (S', X') and $(S', X') \in \text{tpc}\mathcal{BPP}$, then (S, X) is in $\text{tpc}\mathcal{BPP}$.
2. In analogy to Exercise 10.16, prove that randomized reductions preserve solveability by probabilistic algorithms that err with probability at most $1/3$ on each input and typically run in polynomial-time.
3. Prove that randomized reductions are transitive (cf. Exercise 10.17).
4. Show that the error probability of randomized reductions can be reduced (while preserving the domination condition).

Extend the foregoing to reductions that involve distributional *search* problems.

Exercise 10.22 (simple vs sampleable ensembles – Part 1) Prove that any simple probability ensemble is polynomial-time sampleable.

Guideline: See Exercise 10.14.

Exercise 10.23 (simple vs sampleable ensembles – Part 2) Assuming that $\#\mathcal{P}$ contains functions that are not computable in polynomial-time, prove that there exists polynomial-time sampleable ensembles that are not simple.

Guideline: Consider any $R \in \mathcal{PC}$ and suppose that p is a polynomial such that $(x, y) \in R$ implies $|y| = p(|x|)$. Then consider the sampling algorithm A that, on input 1^n, uniformly selects $(x, y) \in \{0,1\}^{n-1} \times \{0,1\}^{p(n-1)}$ and outputs $x1$ if $(x, y) \in R$ and $x0$ otherwise. Note that $\#R(x) = 2^{p(|x|-1)} \cdot \Pr[A(1^{|x|-1}) = x1]$.

Exercise 10.24 (distributional versions of NPC problems – Part 1 [28]) Prove that for any NP-complete problem S there exists a polynomial-time sampleable ensemble X such that any problem in $\text{dist}\mathcal{NP}$ is reducible to (S, X). We actually assume that the many-to-one reductions establishing the NP-completeness of S do not shrink the length of the input.

Guideline: Prove that the guaranteed reduction of $S_{\mathbf{u}}$ to S also reduces $(S_{\mathbf{u}}, U')$ to (S, X), for some sampleable probability ensemble X. Consider first the case that the standard reduction of $S_{\mathbf{u}}$ to S is length preserving, and prove that, when applied to a sampleable probability ensemble, it induces a sampleable distribution on the instances of S. (Note that U' is sampleable (by Exercise 10.22).) Next extend the treatment to the general case, where applying the standard reduction to U'_n induces a distribution on $\cup_{m=n}^{\text{poly}(n)} \{0,1\}^m$ (rather than a distribution on $\{0,1\}^n$).

Exercise 10.25 (distributional versions of NPC problems – Part 2 [28]) Prove Theorem 10.23 (i.e., for any NP-complete problem S there exists a polynomial-time sampleable ensemble X such that any problem in samp$\mathcal{NP}$ is reducible to (S, X)). As in Exercise 10.24, we actually assume that the many-to-one reductions establishing the NP-completeness of S do not shrink the length of the input.

Guideline: We establish the claim for $S_{\mathtt{u}}$, and the general claim follows by using the reduction of $S_{\mathtt{u}}$ to S (as in Exercise 10.24). Thus, we focus on showing that, for some (suitably chosen) sampleable ensemble X, any $(S', X') \in$ samp$\mathcal{NP}$ is reducible to $(S_{\mathtt{u}}, X)$. Loosely speaking, X will be an adequate convex combination of all sampleable distributions (and thus X will not equal U' or U). Specifically, $X = \{X_n\}_{n\in\mathbb{N}}$ is defined such that X_n uniformly selects $i \in [n]$, emulates the execution of the i^{th} algorithm (in lexicographic order) on input 1^n for n^3 steps,[22] and outputs whatever the latter has output (or 0^n in case the said algorithm has not halted within n^3 steps). Prove that, for any $(S'', X'') \in$ samp$\mathcal{NP}$ such that X'' is sampleable in cubic time, the standard reduction of S'' to $S_{\mathtt{u}}$ reduces (S'', X'') to $(S_{\mathtt{u}}, X)$ (as per Definition 10.15; i.e., in particular, it satisfies the domination condition).[23] Finally, using adequate padding, reduce any $(S', X') \in$ samp$\mathcal{NP}$ to some $(S'', X'') \in$ samp$\mathcal{NP}$ such that X'' is sampleable in cubic time.

Exercise 10.26 (search vs decision in the context of sampleable ensembles) Prove that every problem in samp$\mathcal{NP}$ is reducible to some problem in samp$\mathcal{PC}$, and every problem in samp$\mathcal{PC}$ is *randomly* reducible to some problem in samp$\mathcal{NP}$.

Guideline: See proof of Theorem 10.21.

---

[22]Needless to say, the choice to consider n algorithms in the definition of X_n is quite arbitrary. Any other unbounded function of n that is at most a polynomial (and is computable in polynomial-time) will do. (More generally, we may select the i^{th} algorithm with p_i, as long as p_i is a noticeable function of n.) Likewise, the choice to emulate each algorithm for a cubic number of steps (rather some other fixed polynomial number of steps) is quite arbitrary.

[23]Note that applying this reduction to X'' yields an ensembles that is also sampleable in cubic time. This claim uses the fact that the standard reduction runs in time that is less than cubic (and in fact almost linear) in its output, and the fact that the output is longer than the input.

Appendix D

Probabilistic Preliminaries and Advanced Topics in Randomization

What is this? Chicken Quesadilla and Seafood Salad?
Fine, but in the same plate? This is disgusting!

Johan Håstad at Grendel's, Cambridge (1985)

Summary: This appendix lumps together some preliminaries regarding probability theory and some advanced topics related to the role and use of randomness in computation. Needless to say, each of these appears in a separate section.

The probabilistic preliminaries include our conventions regarding random variables, which are used throughout the book. Also included are overviews of three useful inequalities: Markov Inequality, Chebyshev's Inequality, and Chernoff Bound.

The advanced topics include hashing, sampling, and randomness extraction. For hashing, we describe constructions of pairwise (and t-wise independent) hashing functions, and variants of the Leftover Hashing Lemma (which are used a few times in the main text). We then review the "complexity of sampling": that is, the number of samples and the randomness complexity involved in estimating the average value of an arbitrary function defined over a huge domain. Finally, we provide an overview on the question of extracting almost perfect randomness from sources of weak (or defected) randomness.

D.1 Probabilistic preliminaries

Probability plays a central role in complexity theory (see, for example, Chapters 6–9). We assume that the reader is familiar with the basic notions of probability theory. In this section, we merely present the probabilistic notations that are used throughout the book, and three useful probabilistic inequalities.

D.1.1 Notational Conventions

Throughout the entire book we will refer only to *discrete* probability distributions. Specifically, the underlying probability space will consist of the set of all strings of a certain length ℓ, taken with uniform probability distribution. That is, the sample space is the set of all ℓ-bit long strings, and each such string is assigned probability measure $2^{-\ell}$. Traditionally, *random variables* are defined as functions from the sample space to the reals. Abusing the traditional terminology, we use the term random variable also when referring to functions mapping the sample space into the set of binary strings. We often do not specify the probability space, but rather talk directly about random variables. For example, we may say that X is a random variable assigned values in the set of all strings such that $\Pr[X = 00] = \frac{1}{4}$ and $\Pr[X = 111] = \frac{3}{4}$. (Such a random variable may be defined over the sample space $\{0,1\}^2$, so that $X(11) = 00$ and $X(00) = X(01) = X(10) = 111$.) One important case of a random variable is the output of a randomized process (e.g., a probabilistic polynomial-time algorithm, as in Section 6.1).

All our probabilistic statements refer to (functions of) random variables that are defined beforehand. Typically, we may write $\Pr[f(X) = 1]$, where X is a random variable defined beforehand (and f is a function). An important convention is that *all occurrences of the same symbol in a probabilistic statement refer to the same* (unique) *random variable*. Hence, if $B(\cdot,\cdot)$ is a Boolean expression depending on two variables, and X is a random variable then $\Pr[B(X,X)]$ denotes the probability that $B(x,x)$ holds when x is chosen with probability $\Pr[X = x]$. For example, for every random variable X, we have $\Pr[X = X] = 1$. We stress that if we wish to discuss the probability that $B(x,y)$ holds when x and y are chosen independently with identical probability distribution, then we will define *two* independent random variables each with the same probability distribution. Hence, if X and Y are two independent random variables then $\Pr[B(X,Y)]$ denotes the probability that $B(x,y)$ holds when the pair (x,y) is chosen with probability $\Pr[X = x] \cdot \Pr[Y = y]$. For example, for every two independent random variables, X and Y, we have $\Pr[X = Y] = 1$ only if both X and Y are trivial (i.e., assign the entire probability mass to a single string).

Throughout the entire book, U_n denotes a random variable uniformly distributed over the set of strings of length n. Namely, $\Pr[U_n = \alpha]$ equals 2^{-n} if $\alpha \in \{0,1\}^n$ and equals 0 otherwise. We will often refer to the distribution of U_n as the uniform distribution (neglecting to qualify that it is uniform over $\{0,1\}^n$). In addition, we will occasionally use random variables (arbitrarily) distributed over $\{0,1\}^n$ or $\{0,1\}^{\ell(n)}$, for some function $\ell:\mathbb{N}\to\mathbb{N}$. Such random variables are typically denoted by X_n, Y_n, Z_n, etc. We stress that in some cases X_n is distributed over $\{0,1\}^n$, whereas in other cases it is distributed over $\{0,1\}^{\ell(n)}$, for some function $\ell(\cdot)$, which is typically a polynomial. We will often talk about probability ensembles, which are infinite sequence of random variables $\{X_n\}_{n\in\mathbb{N}}$ such that each X_n ranges over strings of length bounded by a polynomial in n.

Statistical difference. The statistical distance (a.k.a variation distance) between the random variables X and Y is defined as

$$\frac{1}{2}\cdot\sum_v |\mathsf{Pr}[X=v]-\mathsf{Pr}[Y=v]| \;=\; \max_S\{\mathsf{Pr}[X\in S]-\mathsf{Pr}[Y\in S]\}. \tag{D.1}$$

We say that X is δ-close (resp., δ-far) to Y if the statistical distance between them is at most (resp., at least) δ.

D.1.2 Three Inequalities

The following probabilistic inequalities are very useful. These inequalities refer to random variables that are assigned real values and provide upper-bounds on the probability that the random variable deviates from its expectation.

Markov Inequality. The most basic inequality is Markov Inequality that applies to any random variable with bounded maximum or minimum value. For simplicity, it is stated for random variables that are lower-bounded by zero, and reads as follows: *Let X be a non-negative random variable and v be a non-negative real number. Then*

$$\mathsf{Pr}\left[X\geq v\right]\leq\frac{\mathsf{E}(X)}{v} \tag{D.2}$$

Equivalently, $\mathsf{Pr}[X\geq r\cdot\mathsf{E}(X)]\leq\frac{1}{r}$. The proof amounts to the following sequence.

$$\begin{aligned}\mathsf{E}(X) &= \sum_x \mathsf{Pr}[X=x]\cdot x\\ &\geq \sum_{x<v}\mathsf{Pr}[X=x]\cdot 0+\sum_{x\geq v}\mathsf{Pr}[X=x]\cdot v\\ &= \mathsf{Pr}[X\geq v]\cdot v\end{aligned}$$

Chebyshev's Inequality: Using Markov's inequality, one gets a potentially stronger bound on the deviation of a random variable from its expectation. This bound, called Chebyshev's inequality, is useful when having additional information concerning the random variable (specifically, a good upper bound on its variance). For a random variable X of finite expectation, we denote by $\mathsf{Var}(X)\stackrel{\text{def}}{=}\mathsf{E}[(X-\mathsf{E}(X))^2]$ the variance of X, and observe that $\mathsf{Var}(X)=\mathsf{E}(X^2)-\mathsf{E}(X)^2$. Chebyshev's inequality then reads as follows: *Let X be a random variable, and $\delta>0$. Then*

$$\mathsf{Pr}\left[|X-\mathsf{E}(X)|\geq\delta\right]\;\leq\;\frac{\mathsf{Var}(X)}{\delta^2}. \tag{D.3}$$

Proof: We define a random variable $Y \stackrel{\text{def}}{=} (X - \mathsf{E}(X))^2$, and apply Markov inequality. We get

$$\begin{aligned}\mathsf{Pr}\left[|X - \mathsf{E}(X)| \geq \delta\right] &= \mathsf{Pr}\left[(X - \mathsf{E}(X))^2 \geq \delta^2\right] \\ &\leq \frac{\mathsf{E}[(X - \mathsf{E}(X))^2]}{\delta^2}\end{aligned}$$

and the claim follows. ■

Corollary (Pairwise Independent Sampling)**:** Chebyshev's inequality is particularly useful in the analysis of the error probability of approximation via repeated sampling. It suffices to assume that the samples are picked in a pairwise independent manner, where $X_1, X_2, ..., X_n$ are pairwise independent if for every $i \neq j$ and every α, β it holds that $\mathsf{Pr}[X_i = \alpha \wedge X_j = \beta] = \mathsf{Pr}[X_i = \alpha] \cdot \mathsf{Pr}[X_j = \beta]$. The corollary reads as follows: *Let $X_1, X_2, ..., X_n$ be pairwise independent random variables with identical expectation, denoted μ, and identical variance, denoted σ^2. Then, for every $\varepsilon > 0$, it holds that*

$$\mathsf{Pr}\left[\left|\frac{\sum_{i=1}^n X_i}{n} - \mu\right| \geq \varepsilon\right] \leq \frac{\sigma^2}{\varepsilon^2 n}. \tag{D.4}$$

Proof: Define the random variables $\overline{X}_i \stackrel{\text{def}}{=} X_i - \mathsf{E}(X_i)$. Note that the $\overline{X}_i$'s are pairwise independent, and each has zero expectation. Applying Chebyshev's inequality to the random variable $\sum_{i=1}^n \frac{X_i}{n}$, and using the linearity of the expectation operator, we get

$$\begin{aligned}\mathsf{Pr}\left[\left|\sum_{i=1}^n \frac{X_i}{n} - \mu\right| \geq \varepsilon\right] &\leq \frac{\mathsf{Var}\left[\sum_{i=1}^n \frac{X_i}{n}\right]}{\varepsilon^2} \\ &= \frac{\mathsf{E}\left[\left(\sum_{i=1}^n \overline{X}_i\right)^2\right]}{\varepsilon^2 \cdot n^2}\end{aligned}$$

Now (again using the linearity of expectation)

$$\mathsf{E}\left[\left(\sum_{i=1}^n \overline{X}_i\right)^2\right] = \sum_{i=1}^n \mathsf{E}\left[\overline{X}_i^2\right] + \sum_{1 \leq i \neq j \leq n} \mathsf{E}\left[\overline{X}_i \overline{X}_j\right]$$

By the pairwise independence of the $\overline{X}_i$'s, we get $\mathsf{E}[\overline{X}_i \overline{X}_j] = \mathsf{E}[\overline{X}_i] \cdot \mathsf{E}[\overline{X}_j]$, and using $\mathsf{E}[\overline{X}_i] = 0$, we get

$$\mathsf{E}\left[\left(\sum_{i=1}^n \overline{X}_i\right)^2\right] = n \cdot \sigma^2$$

The corollary follows. ■

Chernoff Bound: When using pairwise independent sample points, the error probability in the approximation is decreasing linearly with the number of sample points (see Eq. (D.4)). When using totally independent sample points, the error probability in the approximation can be shown to decrease exponentially with the number of sample points. (The random variables $X_1, X_2, ..., X_n$ are said to be totally independent if for every sequence $a_1, a_2, ..., a_n$ it holds that $\mathsf{Pr}[\wedge_{i=1}^n X_i = a_i] = \prod_{i=1}^n \mathsf{Pr}[X_i = a_i]$.) Probability bounds supporting the foregoing statement are given next. The first bound, commonly referred to as Chernoff Bound, concerns 0-1 random variables (i.e., random variables that are assigned as values either 0 or 1), and asserts the following. *Let $p \leq \frac{1}{2}$, and $X_1, X_2, ..., X_n$ be independent 0-1 random variables such that $\mathsf{Pr}[X_i = 1] = p$, for each i. Then, for every $\varepsilon \in (0, p(1-p)]$, we have*

$$\mathsf{Pr}\left[\left|\frac{\sum_{i=1}^n X_i}{n} - p\right| > \varepsilon\right] \;<\; 2 \cdot e^{-\frac{\varepsilon^2}{2p(1-p)} \cdot n} \;\leq\; 2 \cdot e^{-2\varepsilon^2 n} \tag{D.5}$$

Proof Sketch: We upper-bound $\mathsf{Pr}[\sum_{i=1}^n X_i - pn > \varepsilon n]$, and $\mathsf{Pr}[pn - \sum_{i=1}^n X_i > \varepsilon n]$ is bounded similarly. Letting $\overline{X}_i \stackrel{\text{def}}{=} X_i - \mathsf{E}(X_i)$, we apply Markov Inequality to the random variable $e^{\lambda \sum_{i=1}^n \overline{X}_i}$, where $\lambda > 0$ is determined to optimize the expressions that we derive (hint: $\lambda = \Theta(\varepsilon/p(1-p))$ will do). Thus, $\mathsf{Pr}[\sum_{i=1}^n \overline{X}_i > \varepsilon n]$ is upper-bounded by

$$\frac{\mathsf{E}[e^{\lambda \sum_{i=1}^n \overline{X}_i}]}{e^{\lambda \varepsilon n}} = e^{-\lambda \varepsilon n} \cdot \prod_{i=1}^n \mathsf{E}[e^{\lambda \overline{X}_i}]$$

where the equality is due to the independence of the random variables. To simplify the rest of the proof, we establish a sub-optimal bound as follows. Using a Taylor expansion of e^x (e.g., $e^x < 1 + x + x^2$ for $x \leq 1$) and observing that $\mathsf{E}[\overline{X}_i] = 0$, we get $\mathsf{E}[e^{\lambda \overline{X}_i}] < 1 + \lambda^2 \mathsf{E}[\overline{X}_i^2]$, which equals $1 + \lambda^2 p(1-p)$. Thus, $\mathsf{Pr}[\sum_{i=1}^n X_i - pn > \varepsilon n]$ is upper-bounded by $e^{-\lambda \varepsilon n} \cdot (1 + \lambda^2 p(1-p))^n < \exp(-\lambda \varepsilon n + \lambda^2 p(1-p) n)$, which is optimized at $\lambda = \varepsilon/(2p(1-p))$ yielding $\exp(-\frac{\varepsilon^2}{4p(1-p)} \cdot n)$. Needless to say, this method can be applied in more general settings (e.g., for $X_i \in [0,1]$ rather than $X_i \in \{0,1\}$). $\square$

A more general bound, which refers to independent copies of a general (bounded) random variable, is given next (and is commonly referred to as Hoefding Inequality).[1] *Let $X_1, X_2, ..., X_n$ be n independent random variables with identical probability distribution, each ranging over the (real) interval $[a, b]$, and let μ denote the expected value of each of these variables. Then, for every $\varepsilon > 0$,*

$$\mathsf{Pr}\left[\left|\frac{\sum_{i=1}^n X_i}{n} - \mu\right| > \varepsilon\right] < 2 \cdot e^{-\frac{2\varepsilon^2}{(b-a)^2} \cdot n} \tag{D.6}$$

Hoefding Inequality is useful in estimating the average value of a function defined over a large set of values, especially when the desired error probability needs to

[1] A more general form requires the X_i's to be independent, but not necessarily identical, and uses $\mu \stackrel{\text{def}}{=} \frac{1}{n} \sum_{i=1}^n \mathsf{E}(X_i)$. See [10, Apdx. A].

be negligible (i.e., decrease faster than any polynomial in the relevant parameter). Such an estimate can be obtained provided that we can efficiently sample the set and have a bound on the possible values (of the function).

Pairwise independent versus totally independent sampling. Referring to Eq. (D.6), consider, for simplicity, the case that $a = 0 < \mu < b = 1$. In this case, n *independent* samples give an approximation that deviates by ε from the expect value (i.e., μ) with probability, denoted δ, that is exponentially decreasing with $\varepsilon^2 n$. Such an approximation is called an (ε, δ)-approximation, and can be achieved using $n = O(\varepsilon^{-2} \cdot \log(1/\delta))$ sample points. Thus, the number of sample points is polynomially related to ε^{-1} and logarithmically related to δ^{-1}. In contrast, by Eq. (D.4), an (ε, δ)-approximation by n *pairwise independent* samples calls for setting $n = O(\varepsilon^{-2} \cdot \delta^{-1})$. We stress that, *in both cases the number of samples is polynomially related to the desired accuracy of the estimation* (i.e., ε). *The only advantage of totally independent samples over pairwise independent ones is in the dependency of the number of samples on the error probability* (i.e., δ).

D.2 Hashing

Hashing is extensively used in complexity theory. The typical application is mapping arbitrary (unstructured) sets "almost uniformly" to a structured set of adequate size. Specifically, hashing is supposed to map an arbitrary 2^m-subset of $\{0,1\}^n$ to $\{0,1\}^m$ in an "almost uniform" manner.

For a fixed set S of cardinality 2^m, a 1-1 mapping $f_S : S \to \{0,1\}^m$ does exist, but it is not necessarily an efficient one (e.g., it may require "knowing" the entire set S). Clearly, no single function $f : \{0,1\}^n \to \{0,1\}^m$ can map each 2^m-subset of $\{0,1\}^n$ to $\{0,1\}^m$ in a 1-1 manner (or even approximately so). However, a random function $f : \{0,1\}^n \to \{0,1\}^m$ has the property that, for every 2^m-subset $S \subset \{0,1\}^n$, with overwhelmingly high probability f maps S to $\{0,1\}^m$ such that no point in the range has too many f-preimages in S. The problem is that a truly random function is unlikely to have a succinct representation (let alone an efficient evaluation algorithm). We thus seek families of functions that have a similar property, but do have a succinct representation as well as an efficient evaluation algorithm.

D.2.1 Definitions

Motivated by the foregoing discussion, we consider families of functions $\{H_n^m\}_{m<n}$ Such that the following properties hold:

1. For every $S \subset \{0,1\}^n$, with high probability, a function h selected uniformly in H_n^m maps S to $\{0,1\}^m$ in an "almost uniform" manner. For example, we may require that, for any $|S| = 2^m$ and each point y, with high probability over the choice of h, it holds that $|\{x \in S : h(x) = y\}| \leq \mathrm{poly}(n)$.

2. The functions in H_n^m have succinct representation. For example, we may require that $H_n^m \equiv \{0,1\}^{\ell(n,m)}$, for some polynomial ℓ.

3. The functions in H_n^m can be efficiently evaluated. That is, there exists a polynomial-time algorithm that, on input a representation of a function, h (in H_n^m), and a string $x \in \{0,1\}^n$, returns $h(x)$. In some cases we make even more stringent requirements regarding the algorithm (e.g., that it runs in linear space).

Condition 1 was left vague on purpose. At the very least, we require that the expected size of $\{x \in S : h(x) = y\}$ equals $|S|/2^m$. We shall see (in Section D.2.3) that different interpretations of Condition 1 are satisfied by different families of hashing functions. We focus on t-wise independent hashing functions, defined next.

Definition D.1 (t-wise independent hashing functions): *A family H_n^m of functions from n-bit strings to m-bit strings is called* t-wise independent *if for every t distinct domain elements $x_1, ..., x_t \in \{0,1\}^n$ and every $y_1, ..., y_t \in \{0,1\}^m$ it holds that*

$$\mathsf{Pr}_{h \in H_n^m}[\wedge_{i=1}^t h(x_i) = y_i] \;=\; 2^{-t \cdot m}$$

That is, a uniformly chosen $h \in H_n^m$ maps every t domain elements to the range in a totally uniform manner. Note that for $t \geq 2$, it follows that the probability that a random $h \in H_n^m$ maps two distinct domain elements to the same image equals 2^{-m}. Such (families of) functions are called universal (cf. [47]), but we will focus on the stronger condition of t-wise independence.

D.2.2 Constructions

The following constructions are merely a re-interpretation of the constructions presented in §8.6.1.1. (Alternatively, one may view the constructions presented in §8.6.1.1 as a re-interpretation of the following two constructions.)

Construction D.2 (t-wise independent hashing): *For $t, m, n \in \mathbb{N}$ such that $m \leq n$, consider the following family of hashing functions mapping n-bit strings to m-bit strings. Each t-sequence $\overline{s} = (s_0, s_1, ..., s_{t-1}) \in \{0,1\}^{t \cdot n}$ describes a function $h_{\overline{s}} : \{0,1\}^n \to \{0,1\}^m$ such that $h_{\overline{s}}(x)$ equals the m-bit prefix of the binary representation of $\sum_{j=0}^{t-1} s_j x^j$, where the arithmetic is that of* $\mathrm{GF}(2^n)$*, the finite field of 2^n elements.*

Proposition 8.24 implies that Construction D.2 constitutes a family of t-wise independent hash functions. Typically, we will use either $t = 2$ or $t = \Theta(n)$. To make the construction totally explicit, we need an explicit representation of $\mathrm{GF}(2^n)$; see details following Proposition 8.24. An alternative construction for the case of $t = 2$ may be obtained analogously to the pairwise independent generator of Proposition 8.25. Recall that a Toeplitz matrix is a matrix with all diagonals being homogeneous; that is, $T = (t_{i,j})$ is a Toeplitz matrix if $t_{i,j} = t_{i+1,j+1}$, for all i, j.

Construction D.3 (Alternative pairwise independent hashing): *For $m \leq n$, consider the family of hashing functions in which each n-by-m Toeplitz matrix T and an m-dimensional vector b describes a function $h_{T,b} : \{0,1\}^n \to \{0,1\}^m$ such that $h_{T,b}(x) = Tx + b$.*

Proposition 8.25 implies that Construction D.3 constitutes a family of pairwise independent hash functions. Note that a n-by-m Toeplitz matrix can be specified by $n+m-1$ bits, yielding a description length of $n+2m-1$ bits. An alternative construction (analogous to Eq. (8.18) and requiring $m \cdot n + m$ bits of representation) uses arbitrary n-by-m matrices rather than Toeplitz matrices.

D.2.3 The Leftover Hash Lemma

We now turn to the "almost uniform" cover condition (i.e., Condition 1) mentioned in Section D.2.1. One concrete interpretation of this condition is given by the following lemma (and another is implied by it: see Theorem D.5).

Lemma D.4 *Let $m \leq n$ be integers, H_n^m be a family of pairwise independent hash functions, and $S \subseteq \{0,1\}^n$. Then, for every $y \in \{0,1\}^m$ and every $\varepsilon > 0$, for all but at most an $\frac{2^m}{\varepsilon^2 |S|}$ fraction of $h \in H_n^m$ it holds that*

$$|\{x \in S : h(x) = y\}| = (1 \pm \varepsilon) \cdot \frac{|S|}{2^m}. \tag{D.7}$$

By pairwise independence (or rather even by "1-wise independence"), the expected size of $\{x \in S : h(x) = y\}$ is $|S|/2^m$, where the expectation is taken uniformly over all $h \in H_n^m$. The lemma upper bounds the fraction of h's that deviate from the expected behavior (i.e., for which $|h^{-1}(y) \cap S| \neq (1 \pm \varepsilon) \cdot |S|/2^m$). Needless to say, the bound is meaningful only in case $|S| > 2^m$ (or alternatively for $\varepsilon > 1$). Setting $\varepsilon = \sqrt[3]{2^m/|S|}$ (and focusing on the case that $|S| > 2^m$), we infer that *for all but at most an ε fraction of $h \in H_n^m$ it holds that* $|\{x \in S : h(x) = y\}| = (1 \pm \varepsilon) \cdot |S|/2^m$. Thus, each range element has approximately the right number of h-preimages in the set S, under almost all $h \in H_n^m$.

Proof: Fixing an arbitrary set $S \subseteq \{0,1\}^n$ and an arbitrary $y \in \{0,1\}^m$, we estimate the probability that a uniformly selected $h \in H_n^m$ violates Eq. (D.7). We define random variables ζ_x, over the aforementioned probability space, such that $\zeta_x = \zeta_x(h)$ equal 1 if $h(x) = y$ and $\zeta_x = 0$ otherwise. The expected value of $\sum_{x \in S} \zeta_x$ is $\mu \stackrel{\text{def}}{=} |S| \cdot 2^{-m}$, and we are interested in the probability that this sum deviates from the expectation. Applying Chebyshev's Inequality, we get

$$\mathsf{Pr}\left[\left|\mu - \sum_{x \in S} \zeta_x\right| > \varepsilon \cdot \mu\right] \;<\; \frac{\mu}{\varepsilon^2 \mu^2}$$

because $\mathsf{Var}[\sum_{x \in S} \zeta_x] < |S| \cdot 2^{-m}$ by the pairwise independence of the ζ_x's and the fact that $\mathsf{E}[\zeta_x] = 2^{-m}$. The lemma follows. ■

A generalization (called mixing). The proof of Lemma D.4 can be easily extended to show that *for every set $T \subset \{0,1\}^m$ and every $\varepsilon > 0$, for all but at most an $\frac{2^m}{|T|\cdot|S|\varepsilon^2}$ fraction of $h \in H_n^m$ it holds that $|\{x \in S : h(x) \in T\}| = (1 \pm \varepsilon) \cdot |T| \cdot |S|/2^m$.* (Hint: redefine $\zeta_x = \zeta(h) = 1$ if $h(x) \in T$ and $\zeta_x = 0$ otherwise.) This assertion is meaningfull provided that $|T| \cdot |S| > 2^m/\varepsilon^2$, and in the case that $m = n$ it is called a mixing property.

An extremely useful corollary. The aforementioned generalization of Lemma D.4 asserts that most functions behave well with respect to any fixed sets of preimages $S \subset \{0,1\}^n$ and images $T \subset \{0,1\}^m$. A seemingly stronger statement, which is (non-trivially) implied by Lemma D.4 itself, is that for all adequate sets S most functions $h \in H_n^m$ map S to $\{0,1\}^m$ in an almost uniform manner.[2] This is a consequence of the following theorem.

Theorem D.5 (a.k.a Leftover Hash Lemma): *Let H_n^m and $S \subseteq \{0,1\}^n$ be as in Lemma D.4, and define $\varepsilon = \sqrt[3]{2^m/|S|}$. Consider random variable X and H that are uniformly distributed on S and H_n^m, respectively. Then, the statistical distance between $(H, H(X))$ and (H, U_m) is at most 2ε.*

Using the terminology of Section D.4, we say that H_n^m yields a strong extractor (with parameters to be spelled out there).

Proof: Let V denote the set of pairs (h, y) that violate Eq. (D.7), and $\overline{V} \stackrel{\text{def}}{=} (H_n^m \times \{0,1\}^m) \setminus V$. Then for every $(h, y) \in \overline{V}$ it holds that

$$\begin{aligned} \mathsf{Pr}[(H, H(X)) = (h, y)] &= \mathsf{Pr}[H = h] \cdot \mathsf{Pr}[h(X) = y] \\ &= (1 \pm \varepsilon) \cdot \mathsf{Pr}[(H, U_m) = (h, y)]. \end{aligned}$$

On the other hand, by Lemma D.4 (which asserts $\mathsf{Pr}[(H, y) \in V] \leq \varepsilon$ for every $y \in \{0,1\}^m$) and the setting of ε, we have $\mathsf{Pr}[(H, U_m) \in V] \leq \varepsilon$. It follows that

$$\begin{aligned} \mathsf{Pr}[(H, H(X)) \in V] &= 1 - \mathsf{Pr}[(H, H(X)) \in \overline{V}] \\ &\leq 1 - \mathsf{Pr}[(H, U_m)) \in \overline{V}] + \varepsilon \;\leq\; 2\varepsilon. \end{aligned}$$

Using all these upper-bounds, we upper-bounded the statistical difference between $(H, H(X))$ and (H, U_m), denoted Δ, by separating the contribution of V and $\overline{V}$. Specifically, we have

$$\begin{aligned} \Delta &= \frac{1}{2} \cdot \sum_{(h,y) \in H_n^m \times \{0,1\}^m} |\mathsf{Pr}[(H, H(X)) = (h, y)] - \mathsf{Pr}[(H, U_m) = (h, y)]| \\ &\leq \frac{\varepsilon}{2} + \frac{1}{2} \cdot \sum_{(h,y) \in V} |\mathsf{Pr}[(H, H(X)) = (h, y)] - \mathsf{Pr}[(H, U_m) = (h, y)]| \\ &\leq \frac{\varepsilon}{2} + \frac{1}{2} \cdot \sum_{(h,y) \in V} (\mathsf{Pr}[(H, H(X)) = (h, y)] + \mathsf{Pr}[(H, U_m) = (h, y)]) \end{aligned}$$

[2] That is, for X and ε as in Theorem D.5 and any $\alpha > 0$, for all but at most an α fraction of the functions $h \in H_n^m$ it holds that $h(X)$ is $(2\varepsilon/\alpha)$-close to U_m.

$$\leq \frac{\varepsilon}{2} + \frac{1}{2} \cdot (2\varepsilon + \varepsilon)$$

and the claim follows. ■

An alternative proof of Theorem D.5. Define the collision probability of a random variable Z, denote $\mathsf{cp}(Z)$, as the probability that two independent samples of Z yield the same result. Alternatively, $\mathsf{cp}(Z) \stackrel{\text{def}}{=} \sum_z \mathsf{Pr}[Z = z]^2$. Theorem D.5 follows by combining the following two facts:

1. A general fact: *If $Z \in [N]$ and $\mathsf{cp}(Z) \leq (1 + 4\epsilon^2)/N$ then Z is ϵ-close to the uniform distribution on $[N]$.*

   We prove the contra-positive: Assuming that the statistical distance between Z and the uniform distribution on $[N]$ equals δ, we show that $\mathsf{cp}(Z) \geq (1+4\delta^2)/N$. This is done by defining $L \stackrel{\text{def}}{=} \{z : \mathsf{Pr}[Z = z] < 1/N\}$, and lower-bounding $\mathsf{cp}(Z)$ by using the fact that the collision probability is minimized on uniform distributions. Specifically, considering the uniform distributions on L and $[N] \setminus L$ respectively, we have

$$\mathsf{cp}(Z) \geq |L| \cdot \left(\frac{\mathsf{Pr}[Z \in L]}{|L|} \right)^2 + (N - |L|) \cdot \left(\frac{\mathsf{Pr}[Z \in [N] \setminus L]}{N - |L|} \right)^2 \qquad \text{(D.8)}$$

   Using $\delta = \rho - \mathsf{Pr}[Z \in L]$, where $\rho = |L|/N$, the r.h.s of Eq. (D.8) equals $1 + \frac{\delta^2}{(1-\rho)\rho} \geq 1 + 4\delta^2$.

2. The collision probability of $(H, H(X))$ is at most $(1 + (2^m/|S|))/(|H_n^m| \cdot 2^m)$. (Furthermore, this holds even if H_n^m is only universal.)

   The proof is by a straightforward calculation. Specifically, note that $\mathsf{cp}(H, H(X)) = |H_n^m|^{-1} \cdot \mathsf{E}_{h \in H_n^m}[\mathsf{cp}(h(X))]$, whereas $\mathsf{E}_{h \in H_n^m}[\mathsf{cp}(h(X))] = |S|^{-2} \sum_{x_1, x_2 \in S} \mathsf{Pr}[H(x_1) = H(x_2)]$. The sum equals $|S| + (|S|^2 - |S|) \cdot 2^{-m}$, and so $\mathsf{cp}(H, H(X)) < |H_n^m|^{-1} \cdot (2^{-m} + |S|^{-1})$.

Note that it follows that $(H, H(X))$ is $\sqrt{2^m/4|S|}$-close to (H, U_m), which is a stronger bound than the one provided in Theorem D.5.

Stronger uniformity via higher independence. Recall that Lemma D.4 asserts that for each point in the range of the hash function, with high probability over the choice of the hash function, *this fixed point* has approximately the expected number of preimages in S. A stronger condition asserts that, with high probability over the choice of the hash function, *every point in its range* has approximately the expected number of preimages in S. Such a guarantee can be obtained when using n-wise independent hashing functions.

Lemma D.6 *Let $m \leq n$ be integers, H_n^m be a family of n-wise independent hash functions, and $S \subseteq \{0,1\}^n$. Then, for every $\varepsilon \in (0,1)$, for all but at most an $2^m \cdot (n \cdot 2^m/\varepsilon^2|S|)^{n/2}$ fraction of the functions $h \in H_n^m$, Eq. (D.7) holds for every $y \in \{0,1\}^m$.*

Indeed, the lemma should be used with $2^m < \varepsilon^2|S|/4n$. In particular, using $m = \log_2|S| - \log_2(5n/\varepsilon^2)$ guarantees that with high probability each range elements has $(1 \pm \varepsilon) \cdot |S|/2^m$ preimages in S. Under this setting of parameters $|S|/2^m = 5n/\varepsilon^2$, which is $\text{poly}(n)$ whenever $\varepsilon = 1/\text{poly}(n)$. Needless to say, this guarantee is stronger than the conclusion of Theorem D.5.

Proof: The proof follows the footsteps of the proof of Lemma D.4, taking advantage of the fact that here the random variables (i.e., the ζ_x's) are n-wise independent. For $t = n/2$, this allows using the so-called $2t^{\text{th}}$ *moment analysis*, which generalizes the second moment analysis of pairwise independent sampling (presented in Section D.1.2). As in the proof of Lemma D.4, we fix any S and y, and define $\zeta_x = \zeta_x(h) = 1$ if and only if $h(x) = y$. Letting $\mu = \mathsf{E}[\sum_{x\in S}\zeta_x] = |S|/2^m$ and $\overline{\zeta}_x = \zeta_x - \mathsf{E}(\zeta_x)$, we start with Markov inequality:

$$\begin{aligned}\mathsf{Pr}\left[\left|\mu - \sum_{x\in S}\zeta_x\right| > \varepsilon\cdot\mu\right] &< \frac{\mathsf{E}[(\sum_{x\in S}\overline{\zeta}_x)^{2t}]}{\varepsilon^{2t}\mu^{2t}} \\ &= \frac{\sum_{x_1,\ldots,x_{2t}\in S}\mathsf{E}[\prod_{i=1}^{2t}\overline{\zeta}_{x_i}]}{\varepsilon^{2t}\cdot(|S|/2^m)^{2t}} \qquad \text{(D.9)}\end{aligned}$$

Using $2t$-wise independence, we note that only the terms in Eq. (D.9) that do not vanish are those in which each variable appears with multiplicity. This mean that only terms having less than t distinct variables contribute to Eq. (D.9). Now, for every $j \le t$, we have less than $\binom{|S|}{j}\cdot(2t!) < (2t!/j!)\cdot|S|^j$ terms with j distinct variables, and each such term contributes less than $(2^{-m})^j$ to the sum. Thus, Eq. (D.9) is upper-bounded by

$$\frac{2t!}{(\varepsilon|S|/2^m)^{2t}}\cdot\sum_{j=1}^{t}\frac{(|S|/2^m)^j}{j!} < 2\cdot\frac{2t!/t!}{(\varepsilon^2|S|/2^m)^t} < \left(\frac{2t\cdot 2^m}{\varepsilon^2|S|}\right)^t$$

where the first inequality assumes $|S| > n2^m$ (since the claim hold vacuously otherwise). This upper-bounds the probability that a random $h \in H_n^m$ violates Eq. (D.7) with reprect to a fixed y. Using a union bound on all $y \in \{0,1\}^m$, the lemma follows. ■

D.3 Sampling

In many settings repeated sampling is used to estimate the average of a huge set of values. Namely, given a "value" function $\nu:\{0,1\}^n \to \mathbb{R}$, one wishes to approximate $\bar{\nu} \stackrel{\text{def}}{=} \frac{1}{2^n}\sum_{x\in\{0,1\}^n}\nu(x)$ without having to inspect the value of ν at each point of the domain. The obvious thing to do is sampling the domain at random, and obtaining an approximation to $\bar{\nu}$ by taking the average of the values of ν on the sample points. It turns out that certain "pseudorandom" sequences of sample points may serve almost as well as truly random sequences of sample points, and thus the current problem is indeed related to Section 8.6.

D.3.1 Formal Setting

It is essential to have the range of ν be bounded (or else no reasonable approximation is possible). For simplicity, we adopt the convention of having $[0,1]$ be the range of ν, and the problem for other (predetermined) ranges can be treated analogously. Our notion of approximation depends on two parameters: accuracy (denoted ε) and error probability (denoted δ). We wish to have an algorithm that, with probability at least $1-\delta$, gets within ε of the correct value. This leads to the following definition.

Definition D.7 (sampler): *A* sampler *is a randomized algorithm that on input parameters* n (length)*,* ε (accuracy) *and* δ (error)*, and oracle access to* any *function* $\nu:\{0,1\}^n \rightarrow [0,1]$*, outputs, with probability at least* $1-\delta$*, a value that is at most* ε *away from* $\bar{\nu} \stackrel{\rm def}{=} \frac{1}{2^n}\sum_{x\in\{0,1\}^n} \nu(x)$*. Namely,*

$$\mathsf{Pr}[|\text{sampler}^{\nu}(n,\varepsilon,\delta) - \bar{\nu}| > \varepsilon] \; < \; \delta$$

where the probability is taken over the internal coin tosses of the sampler, also called its random seed.

A non-adaptive sampler *is a sampler that consists of two deterministic algorithms: a* sample generating *algorithm,* G*, and a* evaluation algorithm, V*. On input* n,ε,δ *and a random* seed *of adequate length, algorithm* G *generates a sequence of queries, denoted* $s_1,...,s_m \in \{0,1\}^n$*. Algorithm* V *is given the corresponding* ν*-values (i.e.,* $\nu(s_1),...,\nu(s_m)$*) and outputs an estimate to* $\bar{\nu}$*.*

We are interested in "the complexity of sampling" quantified as a function of the parameters n, ε and δ. Specifically, we will consider three complexity measures: The sample complexity (i.e., the number of oracle queries made by the sampler); the randomness complexity (i.e., the length of the random seed used by the sampler); and the computational complexity (i.e., the running-time of the sampler). We say that a sampler is efficient if its running-time is polynomial in the total length of its queries (i.e., polynomial in both its sample complexity and in n). We will focus on efficient samplers. Furthermore, we will focus on efficient samplers that have optimal (up-to a constant factor) sample complexity, and will wish the randomness complexity to be as low as possible.

D.3.2 Known Results

We note that all the following positive results refer to non-adaptive samplers, whereas the lower bound hold also for general samplers. For more details on these results, see [86, Sec. 3.6.4] and the references therein.

The naive sampler. The straightforward method (or the naive sampler) consists of *uniformly and independently* selecting sufficiently many sample points (queries), and outputting the average value of the function on these points. Using Chernoff Bound it follows that $O(\frac{\log(1/\delta)}{\varepsilon^2})$ sample points suffice. As indicated next, the naive sampler is optimal (up-to a constant factor) in its sample complexity, but is quite wasteful in randomness.

It is known that $\Omega(\frac{\log(1/\delta)}{\varepsilon^2})$ samples are needed in any sampler, and that that samplers that make $s(n,\varepsilon,\delta)$ queries require randomness at least $n+\log_2(1/\delta)-\log_2 s(n,\varepsilon,\delta)-O(1)$. These lower bounds are tight (as demonstrated by non-explicit and inefficient samplers). These facts guide our quest for improvements, which is aimed at finding more randomness-efficient ways of *efficiently* generating sample sequences that can be used in conjunction with an appropriate evaluation algorithm V. (We stress that V need not necessarily take the average of the values of the sampled points.)

The pairwise-independent sampler. Using a pairwise-independence generator (cf. §8.6.1.1) for generating sample points, along with the natural evaluation algorithm (which outputs the average of the values of these points), we can obtain a great saving in the randomness complexity: In particular, using a seed of length $2n$, we can generate $O(1/\delta\varepsilon^2)$ pairwise-independent sample points, which (by Eq. (D.4)) suffice for getting accuracy ε with error δ. Thus, this (Pairwise-Independent) sampler uses $2n$ random bits rather than the $\Omega((\log(1/\delta))\varepsilon^{-2}\cdot n)$ coins used by the naive sampler. Furthermore, for constant $\delta>0$, the Pairwise-Independent Sampler is optimal up-to a constant factor in both its sample and randomness complexities. However, for small δ (i.e., $\delta=o(1)$), this sampler is wasteful in sample complexity.

The Median-of-Averages sampler. A new idea is required for going further, and a relevant tool – random walks on expander graphs (see Sections 8.6.3 and E.2) – is needed too. Specifically, we combine the Pairwise-Independent Sampler with the Expander Random Walk Generator (see Proposition 8.29) to obtain a new sampler. The new sampler uses a t-long random walk on an expander with vertex set $\{0,1\}^{2n}$ for *generating a sequence of* $t\stackrel{\rm def}{=}O(\log(1/\delta))$ *related seeds for* t *invocations of the Pairwise-Independent Sampler*, where each of these invocations uses the corresponding $2n$ bits to generate a sequence of $O(1/\varepsilon^2)$ samples in $\{0,1\}^n$. Furthermore, each of these invocations returns a value that, with probability at least 0.9, is ε-close to $\bar{\nu}$. Theorem 8.28 (see also Exercise 8.36) is used to show that, with probability at least $1-\exp(-t)=1-\delta$, *most* of these t invocations return an ε-close approximation. Hence, *the median among these* t *values is an* (ε,δ)*-approximation to the correct value.* The resulting sampler, called the Median-of-Averages Sampler, has sample complexity $O(\frac{\log(1/\delta)}{\varepsilon^2})$ and randomness complexity $2n+O(\log(1/\delta))$, which is optimal up-to a constant factor in both complexities.

Further improvements. The randomness complexity of the Median-of-Averages Sampler can be improved from $2n+O(\log(1/\delta))$ to $n+O(\log(1/\delta\varepsilon))$, while maintaining its (optimal) sample complexity (of $O(\frac{\log(1/\delta)}{\varepsilon^2})$). This is done by replacing the Pairwise Independent Sampler by a sampler that picks a random vertex in a suitable expander and samples all its neighbors.

Averaging Samplers. Averaging (a.k.a. "Oblivious") samplers are non-adaptive samplers in which the evaluation algorithm is the natural one: that is, it merely outputs the average of the values of the sampled points. Indeed, the Pairwise-Independent Sampler is an averaging sampler, whereas the Median-of-Averages Sampler is not. Interestingly, averaging samplers have applications for which ordinary non-adaptive samplers do not suffice. Averaging samplers are closely related to randomness extractors, defined and discussed in Section D.4.

An odd perspective. Recall that a non-adaptive sampler consists of a sample generator G and an evaluator V such that for every $\nu:\{0,1\}^n \to [0,1]$ it holds that

$$\mathsf{Pr}_{(s_1,...,s_m)\leftarrow G(U_k)}[|V(\nu(s_1),...,\nu(s_m)) - \bar{\nu}| > \varepsilon] \;<\; \delta.$$

Thus, we may view G as a pseudorandom generator that is subjected to a distinguishability test that is determined by a fixed algorithm V and an arbitrary function $\nu:\{0,1\}^n \to [0,1]$, where we assume that $\mathsf{Pr}[|V(\nu(U_n^{(1)}),...,\nu(U_n^{(m)})) - \bar{\nu}| > \varepsilon] \;<\; \delta$. What is a bit odd here is that, except for the case of averaging samplers, the distinguishability test contains a central component (i.e., the evaluator V) that is potentially custom-made to help the generator G pass the test.[3]

D.4 Randomness Extractors

Extracting almost-perfect randomness from sources of weak (i.e., defected) randomness is crucial for the actual use of randomized algorithms, procedures and protocols. The latter are analyzed assuming that they are given access to a perfect random source, while in reality one typically has access only to sources of weak (i.e., highly imperfect) randomness. Randomness extractors are efficient procedures that (possibly with the help of little extra randomness) enhance the quality of random sources, converting any source of weak randomness to an almost perfect one. In addition, randomness extractors are related to several other fundamental problems, to be further discussed later.

One key parameter, which was avoided in the foregoing discussion, is the class of weak random sources from which we need to extract almost perfect randomness. It is preferable to make as little assumptions as possible regarding the weak random source. In other words, we wish to consider a wide class of such sources, and require that the randomness extractor (often referred to as the extractor) "works well" for any source in this class. A general class of such sources is defined in §D.4.1.1, but first we wish to mention that even for very restricted classes of sources no deterministic extractor can work.[4] To overcome this impossibility result, two approaches are used:

[3]Another aspect in which samplers differ from the various pseudorandom generators discussed in Chapter 8 is in the aim to minimize, rather than maximize, the number of blocks (denoted here by m) in the output sequence. However, also in case of samplers the aim is to maximize the block-length (denoted here by n).

[4]For example, consider the class of sources that output n-bit strings such that no string occurs with probability greater than $2^{-(n-1)}$ (i.e., twice its probability weight under the uniform distribution).

Seeded extractors: The first approach consists of considering randomized extractors that use a relatively small amount of randomness (in addition to the weak random source). That is, these extractors obtain two inputs: a short truly random seed and a relatively long sequence generated by an arbitrary source that belongs to the specified class of sources. This suggestion is motivated in two different ways:

1. The application may actually have access to an almost-perfect random source, but bits from this source are much more expensive than bits from the weak (i.e., low-quality) random source. Thus, it makes sense to obtain few high-quality bits from the almost-perfect source and use them to "purify" the cheap bits obtained from the weak (low-quality) source.
2. In some applications (e.g., when using randomized algorithms), it may be possible to scan over all possible values of the seed and run the algorithm using the corresponding extracted randomness. That is, we obtain a sample r from the weak random source, and invoke the algorithm on $\texttt{extract}(s, r)$, for every possible seed s, ruling by majority. (This alternative is typically not applicable to cryptographic and/or distributed settings.)

Few independent sources: The second approach consists of considering deterministic extractors that obtain samples from a few (say two) *independent* sources of weak randomness. Such extractors are applicable in any setting (including in cryptography), provided that the application has access to the required number of independent weak random sources.

In this section we focus on the first type of extractors (i.e., the *seeded extractors*). This choice is motivated both by the relatively more mature state of the research in that direction and the closer connection between this direction and other topics in complexity.

D.4.1 Definitions and various perspectives

We first present a definition that corresponds to the foregoing motivational discussion, and later discuss its relation to other topics in complexity.

D.4.1.1 The Main Definition

A very wide class of weak random sources corresponds to sources for which no specific output is too probable (cf. [52]). That is, the class is parameterized by a (probability) bound β and consists of all sources X such that for every x it holds that $\mathsf{Pr}[X = x] \leq \beta$. In such a case, we say that X has min-entropy[5] at least $\log_2(1/\beta)$. Indeed, we represent sources as random variables, and assume that

[5]Recall that the entropy of a random variable X is defined as $\sum_x \mathsf{Pr}[X = x] \log_2(1/\mathsf{Pr}[X = x])$. Indeed the min-entropy of X equals $\min_x\{\log_2(1/\mathsf{Pr}[X = x])\}$, and is always upper-bounded by its entropy.

they are distributed over strings of a fixed length, denoted n. An (n,k)-source is a source that is distributed over $\{0,1\}^n$ and has min-entropy at least k.

An interesting special case of (n,k)-sources is that of sources that are uniform over a subset of 2^k strings. Such sources are called (n,k)-flat. A simple but useful observation is that each (n,k)-source is a convex combination of (n,k)-flat sources.

Definition D.8 (extractor for (n,k)-sources):

1. *An algorithm* $\text{Ext}:\{0,1\}^d \times \{0,1\}^n \to \{0,1\}^m$ *is called an* extractor with error ε for the class $\mathcal{C}$ *if for every source* X *in* $\mathcal{C}$ *it holds that* $\text{Ext}(U_d,X)$ *is* ε*-close to* U_m. *If* $\mathcal{C}$ *is the class of* (n,k)*-sources then* Ext *is called a* (k,ε)-extractor.

2. *An algorithm* Ext *is called a* strong extractor with error ε for $\mathcal{C}$ *if for every source* X *in* $\mathcal{C}$ *it holds that* $(U_d, \text{Ext}(U_d,X))$ *is* ε*-close to* (U_d,U_m). *A* strong (k,ε)-extractor *is defined analogously.*

Using the "decomposition" of (n,k)-sources to (n,k)-flat sources, it follows that Ext *is a* (k,ε)*-extractor if and only if it is an extractor with error* ε *for the class of* (n,k)*-flat sources.* (A similar claim holds for strong extractors.) Thus, much of the technical analysis is conducted with respect to the class of (n,k)-flat sources. For example, it is easy to see that, for $d = \log_2(n/\varepsilon^2) + O(1)$, there exists a (k,ε)-extractor $\text{Ext} : \{0,1\}^d \times \{0,1\}^n \to \{0,1\}^k$. (The proof is by the Probabilistic Method and uses a union bound on the set of all (n,k)-flat sources.)[6]

We seek, however, explicit extractors; that is, extractors that are implementable by polynomial-time algorithms. We note that the evaluation algorithm of any family of pairwise independent hash functions mapping n-bit strings to m-bit strings constitutes a (strong) (k,ε)-extractor for $\varepsilon = 2^{-(k-m)/2}$ (see the alternative proof of Theorem D.5). However, these extractors necessarily use a long seed (i.e., $d \geq 2m$ must hold (and in fact $d = n+2m-1$ holds in Construction D.3)). In Section D.4.2 we survey constructions of efficient (k,ε)-extractors that obtain logarithmic seed length (i.e., $d = O(\log(n/\varepsilon))$). But before doing so, we provide a few alternative perspectives on extractors.

An important note on logarithmic seed length. The case of logarithmic seed length is of particular importance for a variety of reasons. Firstly, when emulating a randomized algorithm using a defected random source (as in Item 2 of the motivational discussion of seeded extractors), the overhead is exponential in the length of the seed. Thus, the emulation of a generic probabilistic polynomial-time algorithm can be done in polynomial time only if the seed length is logarithmic. Similarly, the applications discussed in §D.4.1.2 and §D.4.1.3 are feasible only if the seed length is logarithmic. Lastly, we note that logarithmic seed length is an absolute lower-bound for (k,ε)-extractors, whenever $n > k + k^{\Omega(1)}$ (and $m \geq 1$ and $\varepsilon < 1/2$).

[6] The probability that a random function $\text{Ext} : \{0,1\}^d \times \{0,1\}^n \to \{0,1\}^k$ is not an extractor with error ε for a fixed (n,k)-flat source is upper-bounded by $2^{2^k} \cdot \exp(-\Omega(2^{d+k}\varepsilon^2))$, which is smaller than $1/\binom{2^n}{2^k}$.

D.4.1.2 Extractors as averaging samplers

There is a close relationship between extractors and averaging samplers (which are mentioned towards the end of Section D.3). We first show that any averaging sampler gives rise to an extractor. Let $G : \{0,1\}^n \to (\{0,1\}^m)^t$ be the sample generating algorithm of an averaging sampler having accuracy ε and error probability δ. That is, G uses n bits of randomness and generates t sample points in $\{0,1\}^m$ such that for every $f : \{0,1\}^m \to [0,1]$ with probability at least $1-\delta$ the average of the f-values of these points is in the interval $[\overline{f} \pm \varepsilon]$, where $\overline{f} \stackrel{\text{def}}{=} \mathsf{E}[f(U_m)]$. Define $\text{Ext} : [t] \times \{0,1\}^n \to \{0,1\}^m$ such that $\text{Ext}(i,r)$ is the i^{th} sample generated by $G(r)$. We shall prove that Ext is a $(k, 2\varepsilon)$-extractor, for $k = n - \log_2(\varepsilon/\delta)$.

Suppose towards the contradiction that there exists a (n,k)-flat source X such that for some $S \subset \{0,1\}^m$ it is the case that $\mathsf{Pr}[\text{Ext}(U_d, X) \in S] > \mathsf{Pr}[U_m \in S] + 2\varepsilon$, where $d = \log_2 t$ and $[t] \equiv \{0,1\}^d$. Define

$$B = \{x \in \{0,1\}^n : \mathsf{Pr}[\text{Ext}(U_d, x) \in S] > (|S|/2^m) + \varepsilon\}.$$

Then, $|B| > \varepsilon \cdot 2^k = \delta \cdot 2^n$. Defining $f(z) = 1$ if $z \in S$ and $f(z) = 0$ otherwise, we have $\overline{f} \stackrel{\text{def}}{=} \mathsf{E}[f(U_m)] = |S|/2^m$. But, for every $r \in B$ the f-average of the sample $G(r)$ is greater than $\overline{f} + \varepsilon$, in contradiction to the hypothesis that the sampler has error probability δ (with respect to accuracy ε).

We now turn to show that extractors give rise to averaging samplers. Let $\text{Ext} : \{0,1\}^d \times \{0,1\}^n \to \{0,1\}^m$ be a (k,ε)-extractor. Consider the sample generation algorithm $G : \{0,1\}^n \to (\{0,1\}^m)^{2^d}$ define by $G(r) = (\text{Ext}(s,r))_{s \in \{0,1\}^d}$. We prove that it corresponds to an averaging sampler with accuracy ε and error probability $\delta = 2^{-(n-k-1)}$.

Suppose towards the contradiction that there exists a function $f : \{0,1\}^m \to [0,1]$ such that for $\delta 2^n = 2^{k+1}$ strings $r \in \{0,1\}^n$ the average f-value of the sample $G(r)$ deviates from $\overline{f} \stackrel{\text{def}}{=} \mathsf{E}[f(U_m)]$ by more than ε. Suppose, without loss of generality, that for at least half of these r's the average is greater than $\overline{f} + \varepsilon$, and let B denote the set of these r's. Then, for X that is uniformly distributed on B and is thus a (n,k)-source, we have

$$\mathsf{E}[f(\text{Ext}(U_d, X))] > \mathsf{E}[f(U_m)] + \varepsilon,$$

which (using $|f(z)| \leq 1$ for every z) contradicts the hypothesis that $\text{Ext}(U_d, X)$ is ε-close to U_m.

D.4.1.3 Extractors as randomness-efficient error-reductions

As may be clear from the foregoing discussion, extractors yield randomness-efficient methods for error-reduction. Indeed, *error-reduction is a special case of the sampling problem*, obtained by considering Boolean functions. Specifically, for a two-sided error decision procedure A, consider the function $f_x : \{0,1\}^{\rho(|x|)} \to \{0,1\}$ such that $f_x(r) = 1$ if $A(x,r) = 1$ and $f_x(r) = 0$ otherwise. Assuming that the probability that A is correct is at least $0.5 + \varepsilon$ (say $\varepsilon = 1/6$), error reduction amounts to providing a sampler with accuracy ε and any desired error probability $\delta \ll \varepsilon$ for the Boolean function f_x. In particular, any (k,ε)-extractor

$\text{Ext} : \{0,1\}^d \times \{0,1\}^n \to \{0,1\}^{\rho(|x|)}$ with $k = n - \log(1/\delta) - 1$ will do, provided 2^d is feasible (e.g., $2^d = \text{poly}(\rho(|x|))$, where $\rho(\cdot)$ represents the randomness complexity of the original algorithm A). The question of interest here is how does n (which represents the randomness complexity of the corresponding sampler) grow as a function of $\rho(|x|)$ and δ.

Error-reduction using the extractor $\text{Ext}:[\text{poly}(\rho(|x|))]\times\{0,1\}^n\to\{0,1\}^{\rho(|x|)}$

	error probability	randomness complexity
original algorithm	$1/3$	$\rho(\lvert x\rvert)$
resulting algorithm	δ (may depend on $\lvert x\rvert$)	n (function of $\rho(\lvert x\rvert)$ and δ)

Jumping ahead (see Part 1 of Theorem D.10), we note that for every $\alpha > 1$, one can obtain $n = O(\rho(|x|)) + \alpha \log_2(1/\delta)$, for any $\delta > 2^{-\text{poly}(\rho(|x|))}$. Note that, for $\delta < 2^{-O(\rho(|x|))}$, this bound on the randomness-complexity of error-reduction is better than the bound of $n = \rho(|x|) + O(\log(1/\delta))$ that is provided (for the reduction of one-sided error) by the Expander Random Walk Generator (of Section 8.6.3), albeit the number of samples here is larger (i.e., $\text{poly}(\rho(|x|)/\delta)$ rather than $O(\log(1/\delta))$).

Mentioning the reduction of *one-sided* error probability, brings us to a corresponding relaxation of the notion of an extractor, which is called a disperser. Loosely speaking, a (k,ε)-disperser is only required to hit (with positive probability) any set of density greater than ε in its image, rather than produce a distribution that is ε-close to uniform.

Definition D.9 (dispersers): *An algorithm* $\text{Dsp} : \{0,1\}^d \times \{0,1\}^n \to \{0,1\}^m$ *is called a* (k,ε)-disperser *if for every* (n,k)*-source* X *the support of* $\text{Dsp}(U_d, X)$ *covers at least* $(1-\varepsilon)\cdot 2^m$ *points. Alternatively, for every set* $S \subset \{0,1\}^m$ *of size greater than* $\varepsilon 2^m$ *it holds that* $\Pr[\text{Dsp}(U_d, X) \in S] > 0$.

Dispersers can be used for the reduction of one-sided error analogously to the use of extractors for the reduction of two-sided error. Specifically, regarding the aforementioned function f_x (and assuming that either $\Pr[f_x(U_{\ell(|x|)}) = 1] > \varepsilon$ or $f_x(U_{\ell(|x|)}) = 0$), we may use any (k,ε)-disperser $\text{Dsp} : \{0,1\}^d \times \{0,1\}^n \to \{0,1\}^{\ell(|x|)}$ in attempt to find a point z such that $f_x(z) = 1$. Indeed, if $\Pr[f_x(U_{\ell(|x|)}) = 1] > \varepsilon$ then $|\{z : (\forall s \in \{0,1\}^d)\, f_x(\text{Dsp}(s,z)) = 0\}| < 2^k$, and thus the one-sided error can be reduced from $1-\varepsilon$ to $2^{-(n-k)}$ while using n random bits.

D.4.1.4 Other perspectives

Extractors and dispersers have an appealing interpretation in terms of bipartite graphs. Starting with dispersers, we view a disperser $\text{Dsp} : \{0,1\}^d \times \{0,1\}^n \to \{0,1\}^m$ as a bipartite graph $G = ((\{0,1\}^n, \{0,1\}^m), E)$ such that $E = \{(x, \text{Dsp}(s,x)) : x \in \{0,1\}^n, s \in \{0,1\}^d\}$. This graph has the property that *any* subset of 2^k vertices on the left (i.e., in $\{0,1\}^n$) has a neighborhood that contains at least a $1-\varepsilon$ fraction of the vertices of the right, which is remarkable in the typical case where d is small (e.g., $d = O(\log n/\varepsilon)$) and $n \gg k \geq m$ whereas $m = \Omega(k)$ (or at least $m = k^{\Omega(1)}$). Furthermore, if Dsp is efficiently computable then this bipartite graph is strongly constructible in the sense that, given a vertex on the left, one can efficiently find all its neighbors. An extractor $\mathrm{Ext} : \{0,1\}^d \times \{0,1\}^n \to \{0,1\}^m$ yields an analogous graph with a even stronger property: the neighborhood multi-set of *any* subset of 2^k vertices on the left covers the vertices on the right in an almost uniform manner.

An odd perspective. In addition to viewing extractors as averaging samplers, which in turn may be viewed within the scope of the pseudorandomness paradigm, we mention here an even more odd perspective. Specifically, randomness extractors may be viewed as randomized (by the seed) algorithms designed on purpose such that to be fooled by any weak random source (but not by an even worse source). Consider a (k,ε)-extractor $\mathrm{Ext} : \{0,1\}^d \times \{0,1\}^n \to \{0,1\}^m$, for say $\varepsilon \leq 1/100$, $m = k = \omega(\log n/\varepsilon)$ and $d = O(\log n/\varepsilon)$, and a potential test T_S, parameterized by a set $S \subset \{0,1\}^m$, such that $\Pr[T_S(x) = 1] = \Pr[\mathrm{Ext}(U_d, x) \in S]$ (i.e., on input $x \in \{0,1\}^n$, the test uniformly selects $s \in \{0,1\}^d$ and outputs 1 if and only if $\mathrm{Ext}(s,x) \in S$). Then, for every (n,k)-source X the test T_S does not distinguish X from U_n (i.e., $\Pr[T_S(X)] = \Pr[T_S(U_n)] \pm 2\varepsilon$, because $\mathrm{Ext}(U_d, X)$ is 2ε-close to $\mathrm{Ext}(U_d, U_n)$ (since each is ε-close to U_m)). On the other hand, for every $(n, k-d-4)$-flat source Y there exists a set S such that T_S distinguish Y from U_n with gap 0.9 (e.g., for S that equals the support of $\mathrm{Ext}(U_d, Y)$, it holds that $\Pr[T_S(Y)] = 1$ and $\Pr[T_S(U_n)] \leq |S| \cdot 2^{-m} + \varepsilon = 2^{-4} + \varepsilon < 0.1$). Furthermore, this class of tests detects as defected, with probability 2/3, any source that has entropy below $(k/4) - d$.[7] Thus, this weird class of tests views each (n,k)-source as "pseudorandom" while detecting sources of lower entropy (e.g., entropy lower than $(k/4) - d$) as non-pseudorandom. Indeed, this perspective stretches the pseudorandomness paradigm quite far.

D.4.2 Constructions

Recall that we seek explicit constructions of extractors; that is, functions $\mathrm{Ext} : \{0,1\}^d \times \{0,1\}^n \to \{0,1\}^m$ that can be computed in polynomial-time. The question, of course, is of parameters; that is, having (k,ε)-extractors with m as large as possible and d as small as possible. We first note that typically[8] $m \leq k + d - (2\log_2(1/\varepsilon) - O(1))$ and $d \geq \log_2((n-k)/\varepsilon^2) - O(1)$ must hold, regardless of explicitness. The aforementioned bounds are in fact tight; that is, there exists (non-explicit) (k,ε)-extractors with $m = k + d - 2\log_2(1/\varepsilon) - O(1)$ and $d = \log_2((n-k)/\varepsilon^2) + O(1)$. The obvious goal is to meet these bounds via explicit constructions.

[7]For any such source Y, the distribution $Z = \mathrm{Ext}(U_d, Y)$ has entropy at most $k/4 = m/4$, and thus is 0.7-far from U_m (and 2/3-far from $\mathrm{Ext}(U_d, U_n)$). The lower-bound on the statistical distance of Z to U_m can be proven by the contra-positive: if Z is δ-close to U_m then its entropy is at least $(1-\delta)\cdot m - 1$ (e.g., by using Fano's inequality, see [60, Thm. 2.11.1]).

[8]That is, for $\varepsilon < 1/2$ and $m > d$.

D.4.2.1 Some known results

Despite tremendous progress on this problem (and occasional claims regarding "optimal" explicit constructions), the ultimate goal was not reached yet. However, we are pretty close. In particular, we have the following.

Theorem D.10 (explicit constructions of extractors): *Explicit (k, ε)-extractors of the form* $\mathrm{Ext} : \{0,1\}^d \times \{0,1\}^n \to \{0,1\}^m$ *exist in the following cases:*

1. *For any constants $\varepsilon, \alpha > 0$, with $d = O(\log n)$ and $m = (1-\alpha) \cdot k$.*
2. *For any constants $\varepsilon, \alpha > 0$, with $d = (1+\alpha) \cdot \log_2 n$ and $m = k/\mathrm{poly}(\log n)$.*
3. *For any $\varepsilon > \exp(-k/\log k)$, with $d = O(\log n/\varepsilon)$ and $m = \Omega(k/\log k)$.*

Part 2 is due to [188], and the other two parts are due to [148], where these works build on previous ones (which are not cited here). We note that, for sake of simplicity, we did not quote the best possible bounds. Furthermore, we did not mention additional incomparable results (which are relevant for different ranges of parameters). In general, it seems that the "last word" has not been said yet: indeed the current results are close to optimal, but this cannot be said about the way that they are achieved. In view of the foregoing, we refrain from trying to provide an overview of the proof of Theorem D.10, and review instead a conceptual insight that opened the door to much of the recent developments in the area.

D.4.2.2 The pseudorandomness connection

We conclude this section with an overview of a fruitful connection between extractors and certain pseudorandom generators. The connection, discovered by Trevisan [209], is surprising in the sense that it goes in a non-standard direction: it transforms certain pseudorandom generators into extractors. As argued throughout this book (most conspicuously at the end of Section 7.1.2), computational objects are typically more complex than the corresponding information theoretical objects. Thus, if pseudorandom generators and extractors are at all related (which was not suspected before [209]) then this relation should not be expected to help in the construction of extractors, which seem an information theoretic object. Nevertheless, the discovery of this relation did yield a breakthrough in the study of extractors.[9]

But before describing the connection, let us wonder for a moment. Just looking at the syntax, we note that pseudorandom generators have a single input (i.e., the seed), while extractors have two inputs (i.e., the n-bit long source and the d-bit long seed). But taking a second look at the Nisan–Wigderson Generator (i.e., the combination of Construction 8.17 with an amplification of worst-case to average-case hardness), we note that this construction can be viewed as taking two inputs: a d-bit long seed and a "hard" predicate on d'-bit long strings (where $d' = \Omega(d)$).[10]

[9] We note that once the connection became better understood, influence started going in the "right" direction: from extractors to pseudorandom generators.

[10] Indeed, to fit the current context, we have modified some notations. In Construction 8.17 the length of the seed is denoted by k and the length of the input for the predicate is denoted by m.

Now, an appealing idea is to use the n-bit long source as a (truth-table) description of a (worse-case) hard predicate (which indeed means setting $n = 2^{d'}$). The key observation is that *even if the source is only weakly random we expect it to represent a predicate that is hard on the worst-case.*

Recall that the aforementioned construction is supposed to yield a pseudorandom generator whenever it starts with a hard predicate. In the current context, where there are no computational restrictions, pseudorandomness is supposed to hold against any (computationally unbounded) distinguisher, and thus here pseudorandomness means being statistically close to the uniform distribution (on strings of the adequate length, denoted ℓ). Intuitively, this makes sense only if the observed sequence is shorter that the amount of randomness in the source (and seed), which is indeed the case (i.e., $\ell < k + d$, where k denotes the min-entropy of the source). Hence, there is hope to obtain a good extractor this way.

To turn the hope into a reality, we need a proof (which is sketched next). Looking again at the Nisan–Wigderson Generator, we note that the proof of indistinguishability of this generator provides a black-box procedure for computing the underlying predicate when given oracle access to any potential distinguisher. Specifically, in the proofs of Theorems 7.19 and 8.18 (which holds for any $\ell = 2^{\Omega(d')}$)[11], this black-box procedure was implemented by a *relatively small circuit* (which depends on the underlying predicate). Hence, this procedure contains relatively little information (regarding the underlying predicate), on top of the observed ℓ-bit long output of the extractor/generator. Specifically, for some fixed polynomial p, the amount of information encoded in the procedure (and thus available to it) is upper-bound by $b \stackrel{\text{def}}{=} p(\ell)$, while the procedure is suppose to compute the underlying predicate correctly on each input. That is, this amount of information is supposed to fully determine the underlying predicate, which in turn is identical to the n-bit long source. Thus, if the source has min-entropy exceeding b, then it cannot be fully determine using only b bits of information. It follows that the foregoing construction constitutes a $(b + O(1), 1/6)$-extractor (outputting $\ell = b^{\Omega(1)}$ bits), where the constant $1/6$ is the one used in the proof of Theorem 8.18 (and the argument holds provided that $b = n^{\Omega(1)}$). Note that this extractor uses a seed of length $d = O(d') = O(\log n)$. The argument can be extended to obtain $(k, \text{poly}(1/k))$-extractors that output $k^{\Omega(1)}$ bits using a seed of length $d = O(\log n)$, provided that $k = n^{\Omega(1)}$.

We note that the foregoing description has only referred to two abstract properties of the Nisan–Wigderson Generator: (1) the fact that this generator uses any worst-case hard predicate as a black-box, and (2) the fact that its analysis uses any distinguisher as a black-box. In particular, we viewed the amplification of worst-case hardness to inapproximability (performed in Theorem 7.19) as part of the construction of the pseudorandom generator. An alternative presentation, which is more self-contained, replaces the amplification step of Theorem 7.19 by a direct argument in the current (information theoretic) context and plugs the resulting predicate directly into Construction 8.17. The advantages of this alternative include using a simpler amplification (since amplification is simpler in the informa-

[11] Recalling that $n = 2^{d'}$, the restriction $\ell = 2^{\Omega(d')}$ implies $\ell = n^{\Omega(1)}$.

tion theoretic setting than in the computational setting), and deriving transparent construction and analysis (which mirror Construction 8.17 and Theorem 8.18, respectively).

The alternative presentation. The foregoing analysis transforms a generic distinguisher into a procedure that computes the underlying predicate correctly on each input, which fully determines this predicate. Hence, an upper-bound on the information available to this procedure yields an upper-bound on the number of possible outcomes of the source that are bad for the extractor. In the alternative presentation, we transforms a generic distinguisher into a procedure that approximates the underlying predicate; that is, the procedure yields a function that is relatively close to the underlying predicate. If the potential underlying predicates are far apart, then this directly yields the desired bound (on the number of bad outcomes that correspond such predicates). Thus, the idea is to encode the n-bit long source by an error correcting code of length $n' = \mathrm{poly}(n)$ and relative distance $0.5 - (1/n)^2$, and use the resulting codeword as a truth-table of a predicate for Construction 8.17. Such codes (coupled with efficient encoding algorithms) do exist (see Section E.1), and the benefit in using them is that each n'-bit long string (determined by the information available to the aforementioned approximation procedure) may be $(0.5 - (1/n))$-close to at most $O(n^2)$ codewords (which correspond to potential predicates). That is, *the resulting extractor converts the n-bit input x into a codeword $x' \in \{0,1\}^{n'}$, viewed as a predicate over $\{0,1\}^{d'}$* (where $d' = \log_2 n'$), *and evaluates this predicate at the ℓ projections of the d-bit long seed, where these projections are determined by the corresponding set system* (i.e., the ℓ-long sequence of d'-subsets of $[d]$). The analysis mirrors the proof of Theorem 8.18, and yields a bound of $2^{O(\ell^2)} \cdot O(n^2)$ on the number of bad outcomes for the source, where $O(\ell^2)$ upper-bounds the information available to the approximation procedure and $O(n^2)$ upper-bounds the number of source-outcomes that when encoded are each $(0.5 - (1/n))$-close to the approximation procedure.

D.4.2.3 Recommended reading

The interested reader is referred to a survey of Shaltiel [187]. This survey contains a comprehensive introduction to the area, including an overview of the ideas that underly the various constructions. In particular, the survey describes the approaches used before the discovery of the pseudorandomness connection, the connection itself (and the constructions that arise from it), and the "third generation" of constructions that followed.

Bibliography

[1] S. Aaronson. Complexity Zoo. A continueously updated web-site at `http://qwiki.caltech.edu/wiki/Complexity_Zoo/`.

[2] 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.

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

[4] M. Ajtai, J. Komlos, E. Szemerédi. Deterministic Simulation in LogSpace. In *19th ACM Symposium on the Theory of Computing*, pages 132–140, 1987.

[5] 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.

[6] N. Alon, L. Babai and A. Itai. A fast and Simple Randomized Algorithm for the Maximal Independent Set Problem. *J. of Algorithms*, Vol. 7, pages 567–583, 1986.

[7] N. Alon and R. Boppana. The monotone circuit complexity of Boolean functions. *Combinatorica*, Vol. 7 (1), pages 1–22, 1987.

[8] N. Alon, E. Fischer, I. Newman, and A. Shapira. A Combinatorial Characterization of the Testable Graph Properties: It's All About Regularity. In *38th ACM Symposium on the Theory of Computing*, to appear, 2006.

[9] N. Alon, O. Goldreich, J. Håstad, R. Peralta. Simple Constructions of Almost k-wise Independent Random Variables. *Journal of Random structures and Algorithms*, Vol. 3, No. 3, (1992), pages 289–304.

[10] N. Alon and J.H. Spencer. *The Probabilistic Method*. John Wiley & Sons, Inc., 1992.

[11] R. Armoni. On the derandomization of space-bounded computations. In the proceedings of *Random98*, Springer-Verlag, Lecture Notes in Computer Science (Vol. 1518), pages 49–57, 1998.

[12] S. Arora. Approximation schemes for NP-hard geometric optimization problems: A survey. *Math. Programming*, Vol. 97, pages 43–69, July 2003.

[13] S. Arora, C. Lund, R. Motwani, M. Sudan and M. Szegedy. Proof Verification and Intractability of Approximation Problems. *Journal of the ACM*, Vol. 45, pages 501–555, 1998. Preliminary version in *33rd FOCS*, 1992.

[14] S. Arora and S. Safra. Probabilistic Checkable Proofs: A New Characterization of NP. *Journal of the ACM*, Vol. 45, pages 70–122, 1998. Preliminary version in *33rd FOCS*, 1992.

[15] H. Attiya and J. Welch: *Distributed Computing: Fundamentals, Simulations and Advanced Topics*. McGraw-Hill, 1998.

[16] L. Babai. Trading Group Theory for Randomness. In *17th ACM Symposium on the Theory of Computing*, pages 421–429, 1985.

[17] L. Babai, L. Fortnow, and C. Lund. Non-Deterministic Exponential Time has Two-Prover Interactive Protocols. *Computational Complexity*, Vol. 1, No. 1, pages 3–40, 1991. Preliminary version in *31st FOCS*, 1990.

[18] L. Babai, L. Fortnow, L. Levin, and M. Szegedy. Checking Computations in Polylogarithmic Time. In *23rd ACM Symposium on the Theory of Computing*, pages 21–31, 1991.

[19] L. Babai, L. Fortnow, N. Nisan and A. Wigderson. BPP has Subexponential Time Simulations unless EXPTIME has Publishable Proofs. *Complexity Theory*, Vol. 3, pages 307–318, 1993.

[20] L. Babai and S. Moran. Arthur-Merlin Games: A Randomized Proof System and a Hierarchy of Complexity Classes. *Journal of Computer and System Science*, Vol. 36, pp. 254–276, 1988.

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

[22] B. Barak. Non-Black-Box Techniques in Crypptography. PhD Thesis, Weizmann Institute of Science, 2004.

[23] W. Baur and V. Strassen. The Complexity of Partial Derivatives. *Theor. Comput. Sci.* 22, pages 317–330, 1983.

[24] P. Beame and T. Pitassi. Propositional Proof Complexity: Past, Present, and Future. In *Bulletin of the European Association for Theoretical Computer Science*, Vol. 65, June 1998, pp. 66–89.

[25] A. Beimel, Y. Ishai, E. Kushilevitz, and J.F. Raymond. Breaking the $O(n^{1/(2k-1)})$ barrier for information-theoretic private information retrieval. In *43rd IEEE Symposium on Foundations of Computer Science*, pages 261–270, 2002.

[26] M. Bellare, O. Goldreich, and E. Petrank. Uniform Generation of NP-witnesses using an NP-oracle. *Information and Computation*, Vol. 163, pages 510–526, 2000.

[27] M. Bellare, O. Goldreich and M. Sudan. Free Bits, PCPs and Non-Approximability – Towards Tight Results. *SIAM Journal on Computing*, Vol. 27, No. 3, pages 804–915, 1998. Extended abstract in *36th FOCS*, 1995.

[28] S. Ben-David, B. Chor, O. Goldreich, and M. Luby. On the Theory of Average Case Complexity. *Journal of Computer and System Science*, Vol. 44 (2), pages 193–219, 1992.

[29] A. Ben-Dor and S. Halevi. In *2nd Israel Symp. on Theory of Computing and Systems*, IEEE Computer Society Press, pages 108-117, 1993.

[30] M. Ben-Or, O. Goldreich, S. Goldwasser, J. Håstad, J. Kilian, S. Micali and P. Rogaway. Everything Provable is Probable in Zero-Knowledge. In *Crypto88*, Springer-Verlag Lecture Notes in Computer Science (Vol. 403), pages 37–56, 1990

[31] M. Ben-Or, S. Goldwasser, J. Kilian and A. Wigderson. Multi-Prover Interactive Proofs: How to Remove Intractability. In *20th ACM Symposium on the Theory of Computing*, pages 113–131, 1988.

[32] M. Ben-Or, S. Goldwasser and A. Wigderson. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation. In *20th ACM Symposium on the Theory of Computing*, pages 1–10, 1988.

[33] E. Ben-Sasson, O. Goldreich, P. Harsha, M. Sudan, and S. Vadhan. Robust PCPs of proximity, Shorter PCPs and Applications to Coding. In *36th ACM Symposium on the Theory of Computing*, pages 1–10, 2004. Full version in *ECCC*, TR04-021, 2004.

[34] E. Ben-Sasson and M. Sudan. Simple PCPs with Poly-log Rate and Query Complexity. *ECCC*, TR04-060, 2004.

[35] L. Berman and J. Hartmanis. On isomorphisms and density of NP and other complete sets. *SIAM Journal on Computing*, Vol. 6 (2), 1977, pages 305–322. Extended abstract in *8th STOC*, 1976.

[36] M. Blum. A Machine-Independent Theory of the Complexity of Recursive Functions. *Journal of the ACM*, Vol. 14 (2), pages 290–305, 1967.

[37] M. Blum and S. Micali. How to Generate Cryptographically Strong Sequences of Pseudo-Random Bits. *SIAM Journal on Computing*, Vol. 13, pages 850–864, 1984. Preliminary version in *23rd FOCS*, 1982.

[38] M. Blum, M. Luby and R. Rubinfeld. Self-Testing/Correcting with Applications to Numerical Problems. *Journal of Computer and System Science*, Vol. 47, No. 3, pages 549–595, 1993.

[39] A. Bogdanov, K. Obata, and L. Trevisan. A lower bound for testing 3-colorability in bounded-degree graphs. In *43rd IEEE Symposium on Foundations of Computer Science*, pages 93–102, 2002.

[40] A. Bogdanov and L. Trevisan. On worst-case to average-case reductions for NP problems. In *Proc. 44th IEEE Symposium on Foundations of Computer Science*, pages 308–317, 2003.

[41] A. Bogdanov and L. Trevisan. Average-case complexity: a survey. In preparation, 2005.

[42] R. Boppana, J. Håstad, and S. Zachos. Does Co-NP Have Short Interactive Proofs? *Information Processing Letters*, 25, May 1987, pages 127-132.

[43] R. Boppana and M. Sipser. The complexity of finite functions. In *Handbook of Theoretical Computer Science: Volume A – Algorithms and Complexity*, J. van Leeuwen editor, MIT Press/Elsevier, 1990, pages 757–804.

[44] A. Borodin. Computational Complexity and the Existence of Complexity Gaps. *Journal of the ACM*, Vol. 19 (1), pages 158–174, 1972.

[45] A. Borodin. On Relating Time and Space to Size and Depth. *SIAM Journal on Computing*, Vol. 6 (4), pages 733–744, 1977.

[46] G. Brassard, D. Chaum and C. Crépeau. Minimum Disclosure Proofs of Knowledge. *Journal of Computer and System Science*, Vol. 37, No. 2, pages 156–189, 1988. Preliminary version by Brassard and Crépeau in *27th FOCS*, 1986.

[47] L. Carter and M. Wegman. Universal Hash Functions. *Journal of Computer and System Science*, Vol. 18, 1979, pages 143–154.

[48] G.J. Chaitin. On the Length of Programs for Computing Finite Binary Sequences. *Journal of the ACM*, Vol. 13, pages 547–570, 1966.

[49] A.K. Chandra, D.C. Kozen and L.J. Stockmeyer. Alternation. *Journal of the ACM*, Vol. 28, pages 114–133, 1981.

[50] D. Chaum, C. Crépeau and I. Damgård. Multi-party unconditionally Secure Protocols. In *20th ACM Symposium on the Theory of Computing*, pages 11–19, 1988.

[51] B. Chor and O. Goldreich. On the Power of Two–Point Based Sampling. *Jour. of Complexity*, Vol 5, 1989, pages 96–106. Preliminary version dates 1985.

[52] B. Chor and O. Goldreich. Unbiased Bits from Sources of Weak Randomness and Probabilistic Communication Complexity. *SIAM Journal on Computing*, Vol. 17, No. 2, pages 230–261, 1988.

[53] A. Church. An Unsolvable Problem of Elementary Number Theory. *Amer. J. of Math.*, Vol. 58, pages 345–363, 1936.

[54] A. Cobham. The Intristic Computational Difficulty of Functions. In *Proc. 1964 Iternational Congress for Logic Methodology and Philosophy of Science*, pages 24–30, 1964.

[55] S.A. Cook. The Complexity of Theorem Proving Procedures. In *3rd ACM Symposium on the Theory of Computing*, pages 151–158, 1971.

[56] S.A. Cook. A overview of Computational Complexity. Turing Award Lecture. *CACM*, Vol. 26 (6), pages 401–408, 1983.

[57] S.A. Cook. A Taxonomy of Problems with Fast Parallel Algorithms. *Information and Control*, Vol. 64, pages 2–22, 1985.

[58] S.A. Cook and R.A. Reckhow. Stephen A. Cook, Robert A. Reckhow: The Relative Efficiency of Propositional Proof Systems. *J. of Symbolic Logic*, Vol. 44 (1), pages 36–50, 1979.

[59] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9, pages 251–280, 1990.

[60] T.M. Cover and G.A. Thomas. *Elements of Information Theory.* John Wiley & Sons, Inc., New-York, 1991.

[61] P. Crescenzi and V. Kann. A compendium of NP Optimization problems. Available at `http://www.nada.kth.se/~viggo/wwwcompendium/`

[62] W. Diffie, and M.E. Hellman. New Directions in Cryptography. *IEEE Trans. on Info. Theory*, IT-22 (Nov. 1976), pages 644–654.

[63] I. Dinur. The PCP Theorem by Gap Amplification. *ECCC*, TR05-046, 2005.

[64] I. Dinur and O. Reingold. Assignment-testers: Towards a combinatorial proof of the PCP-Theorem. In *45th IEEE Symposium on Foundations of Computer Science*, pages 155–164, 2004.

[65] I. Dinur and S. Safra. The importance of being biased. In *34th ACM Symposium on the Theory of Computing*, pages 33–42, 2002.

[66] J. Edmonds. Paths, Trees, and Flowers. *Canad. J. Math.*, Vol. 17, pages 449–467, 1965.

[67] S. Even. *Graph Algorithms.* Computer Science Press, 1979.

[68] S. Even, A.L. Selman, and Y. Yacobi. The Complexity of Promise Problems with Applications to Public-Key Cryptography. *Information and Control*, Vol. 61, pages 159–173, 1984.

[69] U. Feige, S. Goldwasser, L. Lovász and S. Safra. On the Complexity of Approximating the Maximum Size of a Clique. Unpublished manuscript, 1990.

[70] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy. Approximating Clique is almost NP-complete. *Journal of the ACM*, Vol. 43, pages 268–292, 1996. Preliminary version in *32nd FOCS*, 1991.

[71] U. Feige, D. Lapidot, and A. Shamir. Multiple Non-Interactive Zero-Knowledge Proofs Under General Assumptions. *SIAM Journal on Computing*, Vol. 29 (1), pages 1–28, 1999.

[72] U. Feige and A. Shamir. Witness Indistinguishability and Witness Hiding Protocols. In *22nd ACM Symposium on the Theory of Computing*, pages 416–426, 1990.

[73] E. Fischer. The art of uninformed decisions: A primer to property testing. *Bulletin of the European Association for Theoretical Computer Science*, Vol. 75, pages 97–126, 2001.

[74] G.D. Forney. *Concatenated Codes*. MIT Press, Cambridge, MA 1966.

[75] L. Fortnow, R. Lipton, D. van Melkebeek, and A. Viglas. Time-space lower bounds for satisfiability. *Journal of the ACM*, Vol. 52 (6), pages 835–865, November 2005.

[76] L. Fortnow, J. Rompel and M. Sipser. On the power of multi-prover interactive protocols. In *3rd IEEE Symp. on Structure in Complexity Theory*, pages 156–161, 1988. See errata in *5th IEEE Symp. on Structure in Complexity Theory*, pages 318–319, 1990.

[77] S. Fortune. A Note on Sparse Complete Sets. *SIAM Journal on Computing*, Vol. 8, pages 431–433, 1979.

[78] M. Fürer, O. Goldreich, Y. Mansour, M. Sipser, and S. Zachos. On Completeness and Soundness in Interactive Proof Systems. *Advances in Computing Research: a research annual*, Vol. 5 (Randomness and Computation, S. Micali, ed.), pages 429–442, 1989.

[79] M.L. Furst, J.B. Saxe, and M. Sipser. Parity, Circuits, and the Polynomial-Time Hierarchy. *Mathematical Systems Theory*, Vol. 17 (1), pages 13–27, 1984. Preliminary version in *22nd FOCS*, 1981.

[80] O. Gaber and Z. Galil. Explicit Constructions of Linear Size Superconcentrators. *Journal of Computer and System Science*, Vol. 22, pages 407–420, 1981.

[81] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York, 1979.

[82] D. Gillman. A chernoff bound for random walks on expander graphs. In *34th IEEE Symposium on Foundations of Computer Science*, pages 680–691, 1993.

[83] O. Goldreich. *Foundation of Cryptography – Class Notes*. Computer Science Dept., Technion, Israel, Spring 1989. Superseded by [87, 88].

[84] O. Goldreich. A Note on Computational Indistinguishability. *Information Processing Letters*, Vol. 34, pages 277–281, May 1990.

[85] O. Goldreich. Notes on Levin's Theory of Average-Case Complexity. *ECCC*, TR97-058, Dec. 1997.

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

[87] O. Goldreich. *Foundation of Cryptography: Basic Tools*. Cambridge University Press, 2001.

[88] O. Goldreich. *Foundation of Cryptography: Basic Applications*. Cambridge University Press, 2004.

[89] O. Goldreich. Short Locally Testable Codes and Proofs (Survey). *ECCC*, TR05-014, 2005.

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

[91] O. Goldreich, S. Goldwasser, and S. Micali. How to Construct Random Functions. *Journal of the ACM*, Vol. 33, No. 4, pages 792–807, 1986.

[92] O. Goldreich, S. Goldwasser, and A. Nussboim. On the Implementation of Huge Random Objects. In *44th IEEE Symposium on Foundations of Computer Science*, pages 68–79, 2002.

[93] O. Goldreich, S. Goldwasser, and D. Ron. Property testing and its connection to learning and approximation. *Journal of the ACM*, pages 653–750, July 1998.

[94] O. Goldreich and H. Krawczyk. On the Composition of Zero-Knowledge Proof Systems. *SIAM Journal on Computing*, Vol. 25, No. 1, February 1996, pages 169–192. Preliminary version in *17th ICALP*, 1990.

[95] O. Goldreich and L.A. Levin. Hard-core Predicates for any One-Way Function. In *21st ACM Symposium on the Theory of Computing*, pages 25–32, 1989.

[96] O. Goldreich, S. Micali and A. Wigderson. Proofs that Yield Nothing but their Validity or All Languages in NP Have Zero-Knowledge Proof Systems. *Journal of the ACM*, Vol. 38, No. 3, pages 691–729, 1991. Preliminary version in *27th FOCS*, 1986.

[97] O. Goldreich, S. Micali and A. Wigderson. How to Play any Mental Game – A Completeness Theorem for Protocols with Honest Majority. In *19th ACM Symposium on the Theory of Computing,* pages 218–229, 1987.

[98] O. Goldreich, N. Nisan and A. Wigderson. On Yao's XOR-Lemma. *ECCC*, TR95-050, 1995.

[99] O. Goldreich and D. Ron. Property testing in bounded degree graphs. *Algorithmica*, pages 302–343, 2002.

[100] O. Goldreich and D. Ron. A sublinear bipartite tester for bounded degree graphs. *Combinatorica*, Vol. 19 (3), pages 335–373, 1999.

[101] O. Goldreich, R. Rubinfeld and M. Sudan. Learning polynomials with queries: the highly noisy case. *SIAM J. Discrete Math.*, Vol. 13 (4), pages 535–570, 2000.

[102] O. Goldreich, S. Vadhan and A. Wigderson. On interactive proofs with a laconic provers. *Computational Complexity*, Vol. 11, pages 1–53, 2002.

[103] O. Goldreich and A. Wigderson. Computational Complexity. In *The Princeton Companion to Mathematics*, to appear.

[104] S. Goldwasser and S. Micali. Probabilistic Encryption. *Journal of Computer and System Science*, Vol. 28, No. 2, pages 270–299, 1984. Preliminary version in *14th STOC*, 1982.

[105] S. Goldwasser, S. Micali and C. Rackoff. The Knowledge Complexity of Interactive Proof Systems. *SIAM Journal on Computing*, Vol. 18, pages 186–208, 1989. Preliminary version in *17th STOC*, 1985. Earlier versions date to 1982.

[106] S. Goldwasser, S. Micali, and R.L. Rivest. A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks. *SIAM Journal on Computing*, April 1988, pages 281–308.

[107] 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.

[108] S.W. Golomb. *Shift Register Sequences.* Holden-Day, 1967. (Aegean Park Press, revised edition, 1982.)

[109] J. Hartmanis and R.E. Stearns. On the Computational Complexity of of Algorithms. *Transactions of the AMS*, Vol. 117, pages 285–306, 1965.

[110] J. Håstad. Almost Optimal Lower Bounds for Small Depth Circuits. *Advances in Computing Research: a research annual*, Vol. 5 (Randomness and Computation, S. Micali, ed.), pages 143–170, 1989. Extended abstract in *18th STOC*, pages 6–20, 1986.

[111] J. Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. *Acta Mathematica*, Vol. 182, pages 105–142, 1999. Preliminary versions in *28th STOC* (1996) and *37th FOCS* (1996).

[112] J. Håstad. Getting optimal in-approximability results. In *29th ACM Symposium on the Theory of Computing*, pages 1–10, 1997.

[113] J. Håstad, R. Impagliazzo, L.A. Levin and M. Luby. A Pseudorandom Generator from any One-way Function. *SIAM Journal on Computing*, Volume 28, Number 4, pages 1364–1396, 1999. Preliminary versions by Impagliazzo *et. al.* in *21st STOC* (1989) and Håstad in *22nd STOC* (1990).

[114] J. Håstad and S. Khot. Query efficient PCPs with pefect completeness. In *42nd IEEE Symposium on Foundations of Computer Science*, pages 610–619, 2001.

[115] A. Healy, S. Vadhan and E. Viola. Using nondeterminism to amplify hardness. In *36th ACM Symposium on the Theory of Computing*, pages 192–201, 2004.

[116] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.

[117] D. Hochbaum (ed.). *Approximation Algorithms for NP-Hard Problems.* PWS, 1996.

[118] N. Immerman. Nondeterministic Space is Closed Under Complementation. *SIAM Journal on Computing*, Vol. 17, pages 760–778, 1988.

[119] R. Impagliazzo. Hard-core Distributions for Somewhat Hard Problems. In *36th IEEE Symposium on Foundations of Computer Science*, pages 538–545, 1995.

[120] R. Impagliazzo and L.A. Levin. No Better Ways to Generate Hard NP Instances than Picking Uniformly at Random. In *31st IEEE Symposium on Foundations of Computer Science*, pages 812–821, 1990.

[121] R. Impagliazzo and A. Wigderson. P=BPP if E requires exponential circuits: Derandomizing the XOR Lemma. In *29th ACM Symposium on the Theory of Computing*, pages 220–229, 1997.

[122] R. Impagliazzo and A. Wigderson. Randomness vs Time: Derandomization under a Uniform Assumption. *Journal of Computer and System Science*, Vol. 63 (4), pages 672-688, 2001.

[123] R. Impagliazzo and M. Yung. Direct Zero-Knowledge Computations. In *Crypto87*, Springer-Verlag Lecture Notes in Computer Science (Vol. 293), pages 40–51, 1987.

[124] M. Jerrum, A. Sinclair, and E. Vigoda. A Polynomial-Time Approximation Algorithm for the Permanent of a Matrix with Non-Negative Entries. *Journal of the ACM*, Vol. 51 (4), pages 671–697, 2004.

[125] M. Jerrum, L. Valiant, and V.V. Vazirani. Random Generation of Combinatorial Structures from a Uniform Distribution. *Theoretical Computer Science*, Vol. 43, pages 169–188, 1986.

[126] N. Kahale, Eigenvalues and Expansion of Regular Graphs. *Journal of the ACM*, Vol. 42 (5), pages 1091–1106, September 1995.

[127] R. Kannan, H. Venkateswaran, V. Vinay, and A.C. Yao. A Circuit-based Proof of Toda's Theorem. *Information and Computation*, Vol. 104 (2), pages 271–276, 1993.

[128] R.M. Karp. Reducibility among Combinatorial Problems. In *Complexity of Computer Computations*, R.E. Miller and J.W. Thatcher (eds.), Plenum Press, pages 85–103, 1972.

[129] R.M. Karp and R.J. Lipton. Some connections between nonuniform and uniform complexity classes. In *12th ACM Symposium on the Theory of Computing*, pages 302-309, 1980.

[130] R.M. Karp and M. Luby. Monte-Carlo algorithms for enumeration and reliability problems. In *24th IEEE Symposium on Foundations of Computer Science*, pages 56-64, 1983.

[131] R.M. Karp and V. Ramachandran: Parallel Algorithms for Shared-Memory Machines. In *Handbook of Theoretical Computer Science, Vol A: Algorithms and Complexity*, 1990.

[132] M. Karchmer and A. Wigderson. Monotone Circuits for Connectivity Require Super-logarithmic Depth. *SIAM J. Discrete Math.*, Vol. 3 (2), pages 255–265, 1990. Preliminary version in *20th STOC*, 1988.

[133] M.J. Kearns and U.V. Vazirani. *An introduction to Computational Learning Theory.* MIT Press, 1994.

[134] S. Khot and O. Regev. Vertex Cover Might be Hard to Approximate to within $2-\varepsilon$. In *18th IEEE Conference on Computational Complexity*, pages 379–386, 2003.

[135] V.M. Khrapchenko. A method of determining lower bounds for the complexity of Pi-schemes. In *Matematicheskie Zametki* 10 (1),pages 83–92, 1971 (in Russian). English translation in *Mathematical Notes of the Academy of Sciences of the USSR* 10 (1) 1971, pages 474–479.

[136] J. Kilian. A Note on Efficient Zero-Knowledge Proofs and Arguments. In *24th ACM Symposium on the Theory of Computing*, pages 723–732, 1992.

[137] D.E. Knuth. *The Art of Computer Programming*, Vol. 2 (*Seminumerical Algorithms*). Addison-Wesley Publishing Company, Inc., 1969 (first edition) and 1981 (second edition).

[138] A. Kolmogorov. Three Approaches to the Concept of "The Amount Of Information". *Probl. of Inform. Transm.*, Vol. 1/1, 1965.

[139] E. Kushilevitz and N. Nisan. *Communication Complexity.* Cambridge University Press, 1996.

[140] R.E. Ladner. On the Structure of Polynomial Time Reducibility. *Journal of the ACM*, Vol. 22, 1975, pages 155–171.

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

[142] F.T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes.* Morgan Kaufmann Publishers, San Mateo, CA, 1992.

[143] L.A. Levin. Universal Search Problems. *Problemy Peredaci Informacii 9*, pages 115–116, 1973. Translated in *problems of Information Transmission 9*, pages 265–266.

[144] L.A. Levin. Randomness Conservation Inequalities: Information and Independence in Mathematical Theories. *Information and Control*, Vol. 61, pages 15–37, 1984.

[145] L.A. Levin. Average Case Complete Problems. *SIAM Journal on Computing*, Vol. 15, pages 285–286, 1986.

[146] L.A. Levin. Fundamentals of Computing. *SIGACT News*, Education Forum, special 100-th issue, Vol. 27 (3), pages 89–110, 1996.

[147] M. Li and P. Vitanyi. *An Introduction to Kolmogorov Complexity and its Applications.* Springer Verlag, August 1993.

[148] C.-J. Lu, O. Reingold, S. Vadhan, and A. Wigderson. Extractors: optimal up to constant factors. In *35th ACM Symposium on the Theory of Computing*, pages 602–611, 2003.

[149] A. Lubotzky, R. Phillips, and P. Sarnak. Ramanujan Graphs. *Combinatorica*, Vol. 8, pages 261–277, 1988.

[150] M. Luby and A. Wigderson. Pairwise Independence and Derandomization. TR-95-035, International Computer Science Institute (ICSI), Berkeley, 1995. ISSN 1075-4946.

[151] C. Lund, L. Fortnow, H. Karloff, and N. Nisan. Algebraic Methods for Interactive Proof Systems. *Journal of the ACM*, Vol. 39, No. 4, pages 859–868, 1992. Preliminary version in *31st FOCS*, 1990.

[152] F. MacWilliams and N. Sloane. *The theory of error-correcting codes.* North-Holland, 1981.

[153] G.A. Margulis. Explicit Construction of Concentrators. (In Russian.) *Prob. Per. Infor.*, Vol. 9 (4), pages 71–80, 1973. English translation in *Problems of Infor. Trans.*, pages 325–332, 1975.

[154] S. Micali. Computationally Sound Proofs. *SIAM Journal on Computing*, Vol. 30 (4), pages 1253–1298, 2000. Preliminary version in *35th FOCS*, 1994.

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

[156] P.B. Miltersen and N.V. Vinodchandran. Derandomizing Arthur-Merlin Games using Hitting Sets. *Journal of Computational Complexity*, to appear. Preliminary version in *40th FOCS*, 1999.

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

[158] M. Naor. Bit Commitment using Pseudorandom Generators. *Journal of Cryptology*, Vol. 4, pages 151–158, 1991.

[159] J. Naor and M. Naor. Small-bias Probability Spaces: Efficient Constructions and Applications. *SIAM Journal on Computing*, Vol 22, 1993, pages 838–856.

[160] M. Naor and M. Yung. Universal One-Way Hash Functions and their Cryptographic Application. In *21st ACM Symposium on the Theory of Computing*, 1989, pages 33–43.

[161] N. Nisan. Pseudorandom bits for constant depth circuits. *Combinatorica*, Vol. 11 (1), pages 63–70, 1991.

[162] N. Nisan. Pseudorandom Generators for Space Bounded Computation. *Combinatorica*, Vol. 12 (4), pages 449–461, 1992.

[163] N. Nisan. $\mathcal{RL} \subseteq \mathcal{SC}$. *Journal of Computational Complexity*, Vol. 4, pages 1-11, 1994.

[164] N. Nisan and A. Wigderson. Hardness vs Randomness. *Journal of Computer and System Science*, Vol. 49, No. 2, pages 149–167, 1994.

[165] N. Nisan and D. Zuckerman. Randomness is Linear in Space. *Journal of Computer and System Science*, Vol. 52 (1), pages 43–52, 1996.

[166] C.H. Papadimitriou. *Computational Complexity*. Addison Wesley, 1994.

[167] C.H. Papadimitriou and M. Yannakakis. Optimization, Approximation, and Complexity Classes. In *20th ACM Symposium on the Theory of Computing*, pages 229–234, 1988.

[168] N. Pippenger and M.J. Fischer. Relations among complexity measures. *Journal of the ACM*, Vol. 26 (2), pages 361–381, 1979.

[169] E. Post. A Variant of a Recursively Unsolvable Problem. *Bull. AMS*, Vol. 52, pages 264–268, 1946.

[170] M.O. Rabin. Digitalized Signatures. In *Foundations of Secure Computation* (R.A. DeMillo et. al. eds.), Academic Press, 1977.

[171] M.O. Rabin. Digitalized Signatures and Public Key Functions as Intractable as Factoring. MIT/LCS/TR-212, 1979.

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

[173] R. Raz. A Parallel Repetition Theorem. *SIAM Journal on Computing*, Vol. 27 (3), pages 763–803, 1998. Extended abstract in *27th STOC*, 1995.

[174] R. Raz and A. Wigderson. Monotone Circuits for Matching Require Linear Depth. *Journal of the ACM*, Vol. 39 (3), pages 736–744, 1992. Preliminary version in *22nd STOC*, 1990.

[175] A. Razborov. Lower bounds for the monotone complexity of some Boolean functions. In *Doklady Akademii Nauk SSSR*, Vol. 281, No. 4, 1985, pages 798–801. English translation in *Soviet Math. Doklady*, 31, pages 354–357, 1985.

[176] A. Razborov. Lower bounds on the size of bounded-depth networks over a complete basis with logical addition. In *Matematicheskie Zametki*, Vol. 41, No. 4, pages 598–607, 1987. English translation in *Mathematical Notes of the Academy of Sci. of the USSR*, Vol. 41 (4), pages 333–338, 1987.

[177] A.R. Razborov and S. Rudich. Natural Proofs. *Journal of Computer and System Science*, Vol. 55 (1), pages 24–35, 1997.

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

[179] O. Reingold, S. Vadhan, and A. Wigderson. Entropy Waves, the Zig-Zag Graph Product, and New Constant-Degree Expanders and Extractors. *Annals of Mathematics*, Vol. 155 (1), pages 157–187, 2001. Preliminary version in *41st FOCS*, pages 3–13, 2000.

[180] H.G. Rice. Classes of Recursively Enumerable Sets and their Decision Problems. *Trans. AMS*, Vol. 89, pages 25–59, 1953.

[181] R.L. Rivest, A. Shamir and L.M. Adleman. A Method for Obtaining Digital Signatures and Public Key Cryptosystems. *CACM*, Vol. 21, Feb. 1978, pages 120–126.

[182] D. Ron. Property testing. In *Handbook on Randomization, Volume II*, pages 597–649, 2001. (Editors: S. Rajasekaran, P.M. Pardalos, J.H. Reif and J.D.P. Rolim.)

[183] R. Rubinfeld and M. Sudan. Robust characterization of polynomials with applications to program testing. *SIAM Journal on Computing*, Vol. 25 (2), pages 252–271, 1996.

[184] M. Saks and S. Zhou. RSPACE(S) $\subseteq$ DSPACE($S^{3/2}$). In 36th *IEEE Symposium on Foundations of Computer Science*, pages 344–353, 1995.

[185] W.J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *JCSS*, Vol. 4 (2), pages 177-192, 1970.

[186] A. Selman. On the structure of NP. *Notices Amer. Math. Soc.*, Vol. 21 (6), page 310, 1974.

[187] R. Shaltiel. Recent Developments in Explicit Constructions of Extractors. In *Current Trends in Theoretical Computer Science: The Challenge of the New Century, Vol 1: Algorithms and Complexity*, World scietific, 2004. (Editors: G. Paun, G. Rozenberg and A. Salomaa.) Preliminary version in *Bulletin of the EATCS 77*, pages 67–95, 2002.

[188] R. Shaltiel and C. Umans. Simple Extractors for All Min-Entropies and a New Pseudo-Random Generator. In *42nd IEEE Symposium on Foundations of Computer Science*, pages 648–657, 2001.

[189] C.E. Shannon. A Symbolic Analysis of Relay and Switching Circuits. *Trans. American Institute of Electrical Engineers*, Vol. 57, pages 713–723, 1938.

[190] C.E. Shannon. A mathematical theory of communication. *Bell Sys. Tech. Jour.*, Vol. 27, pages 623–656, 1948.

[191] C.E. Shannon. Communication Theory of Secrecy Systems. *Bell Sys. Tech. Jour.*, Vol. 28, pages 656–715, 1949.

[192] A. Shamir. IP = PSPACE. *Journal of the ACM*, Vol. 39, No. 4, pages 869–877, 1992. Preliminary version in *31st FOCS*, 1990.

[193] A. Shpilka. Lower Bounds for Matrix Product. *SIAM Journal on Computing*, pages 1185-1200, 2003.

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

[195] M. Sipser. *Introduction to the Theory of Computation*. PWS Publishing Company, 1997.

[196] R. Smolensky. Algebraic Methods in the Theory of Lower Bounds for Boolean Circuit Complexity. In *19th ACM Symposium on the Theory of Computing* pages 77–82, 1987.

[197] R.J. Solomonoff. A Formal Theory of Inductive Inference. *Information and Control*, Vol. 7/1, pages 1–22, 1964.

[198] 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.

[199] D.A. Spielman. *Advanced Complexity Theory*, Lectures 10 and 11. Notes (by D. Lewin and S. Vadhan), March 1997. Available from `http://www.cs.yale.edu/homes/spielman/AdvComplexity/1998/` as `lect10.ps` and `lect11.ps`.

[200] L.J. Stockmeyer. The Polynomial-Time Hierarchy. *Theoretical Computer Science*, Vol. 3, pages 1–22, 1977.

[201] L. Stockmeyer. The Complexity of Approximate Counting. In *15th ACM Symposium on the Theory of Computing*, pages 118–126, 1983.

[202] V. Strassen. Algebraic Complexity Theory. In *Handbook of Theoretical Computer Science: Volume A – Algorithms and Complexity*, J. van Leeuwen editor, MIT Press/Elsevier, 1990, pages 633–672.

[203] M. Sudan. Decoding of Reed Solomon codes beyond the error-correction bound. *Journal of Complexity*, Vol. 13 (1), pages 180–193, 1997.

[204] M. Sudan. Algorithmic introduction to coding theory. Lecture notes, Available from `http://theory.csail.mit.edu/~madhu/FT01/`, 2001.

[205] , M. Sudan, L. Trevisan, and S. Vadhan. Pseudorandom generators without the XOR Lemma. *Journal of Computer and System Science*, Vol. 62, No. 2, pages 236–266, 2001.

[206] R. Szelepcsenyi. A Method of Forced Enumeration for Nondeterministic Automata. *Acta Informatica*, Vol. 26, pages 279–284, 1988.

[207] S. Toda. PP is as hard as the polynomial-time hierarchy. *SIAM Journal on Computing*, Vol. 20 (5), pages 865–877, 1991.

[208] B.A. Trakhtenbrot. A Survey of Russian Approaches to *Perebor* (Brute Force Search) Algorithms. *Annals of the History of Computing*, Vol. 6 (4), pages 384–398, 1984.

[209] L. Trevisan. Constructions of Near-Optimal Extractors Using Pseudo-Random Generators. In *31st ACM Symposium on the Theory of Computing*, pages 141–148, 1998.

[210] V. Trifonov. An $O(\log n \log\log n)$ Space Algorithm for Undirected st-Connectivity. In *37th ACM Symposium on the Theory of Computing*, pages 623–633, 2005.

[211] C.E. Turing. On Computable Numbers, with an Application to the Entscheidungsproblem. *Proc. Londom Mathematical Soceity*, Ser. 2, Vol. 42, pages 230–265, 1936. A Correction, *ibid.*, Vol. 43, pages 544–546.

[212] C. Umans. Pseudo-random generators for all hardness. *Journal of Computer and System Science*, Vol. 67 (2), pages 419–440, 2003.

[213] S. Vadhan. A Study of Statistical Zero-Knowledge Proofs. PhD Thesis, Department of Mathematics, MIT, 1999. Available from `http://www.eecs.harvard.edu/~salil/papers/phdthesis-abs.html`.

[214] S. Vadhan. An Unconditional Study of Computational Zero Knowledge. In *45th IEEE Symposium on Foundations of Computer Science*, pages 176–185, 2004.

[215] L.G. Valiant. The Complexity of Computing the Permanent. *Theoretical Computer Science*, Vol. 8, pages 189–201, 1979.

[216] L.G. Valiant. A theory of the learnable. *CACM*, Vol. 27/11, pages 1134–1142, 1984.

[217] L.G. Valiant and V.V. Vazirani. NP Is as Easy as Detecting Unique Solutions. *Theoretical Computer Science*, Vol. 47 (1), pages 85–93, 1986.

[218] J. von Neumann, First Draft of a Report on the EDVAC, 1945. Contract No. W-670-ORD-492, Moore School of Electrical Engineering, Univ. of Penn., Philadelphia. Reprinted (in part) in *Origins of Digital Computers: Selected Papers*, Springer-Verlag, Berlin Heidelberg, pages 383–392, 1982.

[219] J. von Neumann, Zur Theorie der Gesellschaftsspiele. *Mathematische Annalen*, 100, pages 295–320, 1928.

[220] I. Wegener. *The Complexity of Boolean Functions*. Wiley-Teubner, 1987.

[221] I. Wegener. *Branching Programs and Binary Decision Diagrams – Theory and Applications*. SIAM Monographs on Discrete Mathematics and Applications, 2000.

[222] A. Wigderson. The amazing power of pairwise independence. In *26th ACM Symposium on the Theory of Computing*, pages 645–647, 1994.

[223] A.C. Yao. Theory and Application of Trapdoor Functions. In *23rd IEEE Symposium on Foundations of Computer Science*, pages 80–91, 1982.

[224] A.C. Yao. Separating the Polynomial-Time Hierarchy by Oracles. In *26th IEEE Symposium on Foundations of Computer Science*, pages 1-10, 1985.

[225] A.C. Yao. How to Generate and Exchange Secrets. In *27th IEEE Symposium on Foundations of Computer Science*, pages 162–167, 1986.

[226] D. Zuckerman. Simulating BPP Using a General Weak Random Source. *Algorithmica*, Vol. 16, pages 367–391, 1996.

[227] D. Zuckerman. Randomness-Optimal Oblivious Sampling. *Journal of Random Structures and Algorithms*, Vol. 11, Nr. 4, December 1997, pages 345–367.