Chapter 9

Probabilistic Proof Systems

A proof is whatever convinces me.

Shimon Even (1935–2004)

The glory attached to the creativity involved in finding proofs makes us forget that it is the less glorified process of verification that gives proofs their value. Conceptually speaking, proofs are secondary to the verification process; whereas technically speaking, proof systems are defined in terms of their verification procedures.

The notion of a verification procedure presumes the notion of computation and furthermore the notion of efficient computation. This implicit stipulation is made explicit in the definition of $\mathcal{NP}$, where efficient computation is associated with deterministic polynomial-time algorithms. However, as argued next, we can gain a lot if we are willing to take a somewhat non-traditional step and allow *probabilistic* verification procedures.

In this chapter, we shall study three types of probabilistic proof systems, called *interactive proofs*, *zero-knowledge proofs*, and *probabilistic checkable proofs*. In each of these three cases, we shall present fascinating results that cannot be obtained when considering the analogous deterministic proof systems.

Summary: The association of efficient procedures with *deterministic* polynomial-time procedures is the basis for viewing NP-proof systems as the canonical formulation of proof systems (with efficient verification procedures). Allowing *probabilistic* verification procedures and, moreover, ruling by statistical evidence gives rise to various types of probabilistic proof systems. Indeed, these probabilistic proof systems carry a probability of error (which is explicitly bounded and can be reduced by successive application of the proof system), yet they offer various advantages over the traditional (deterministic and errorless) proof systems.

Randomized and interactive verification procedures, giving rise to *interactive proof systems*, seem much more powerful than their deterministic

counterparts. In particular, such interactive proof systems exist for any set in $\mathcal{PSPACE} \supseteq \text{co}\mathcal{NP}$ (e.g., for the set of unsatisfied propositional formulae), whereas it is widely believed that some sets in $\text{co}\mathcal{NP}$ do *not* have NP-proof systems (i.e., $\mathcal{NP} \neq \text{co}\mathcal{NP}$). We stress that a "proof" in this context is not a fixed and static object, but rather a randomized (and dynamic) process in which the verifier interacts with the prover. Intuitively, one may think of this interaction as consisting of questions asked by the verifier, to which the prover has to reply convincingly.

Such randomized and interactive verification procedures allow for the meaningful conceptualization of *zero-knowledge proofs*, which are of great theoretical and practical interest (especially in cryptography). Loosely speaking, zero-knowledge proofs are interactive proofs that yield nothing (to the verifier) beyond the fact that the assertion is indeed valid. For example, a zero-knowledge proof that a certain propositional formula is satisfiable does not reveal a satisfying assignment to the formula nor any partial information regarding such an assignment (e.g., whether the first variable can assume the value `true`). Thus, the successful verification of a zero-knowledge proof exhibit an extreme contrast between being convinced of the validity of a statement and learning nothing else (while receiving such a convincing proof). It turns out that, under reasonable complexity assumptions (i.e., assuming the existence of one-way functions), every set in $\mathcal{NP}$ has a zero-knowledge proof system.

NP-proofs can be efficiently transformed into a (redundant) form that offers a trade-off between the number of locations (randomly) examined in the resulting proof and the confidence in its validity. In particular, it is known that any set in $\mathcal{NP}$ has an NP-proof system that supports probabilistic verification such that the error probability decreases exponentially with the number of bits read from the alleged proof. These redundant NP-proofs are called *probabilistically checkable proofs* (or PCPs). In addition to their conceptually fascinating nature, PCPs are closely related to the study of the complexity of numerous natural approximation problems.

Introduction and Preliminaries

Conceptually speaking, proofs are secondary to the verification process. Indeed, both in mathematics and in real-life, proofs are meaningful only with respect to commonly agreed principles of reasoning, and the verification process amounts to checking that these principles were properly applied. Thus, these principles, which are typically taken for granted, are more fundamental than any specific proof that applies them; that is, the mere attempt to reason about anything is based on commonly agreed principles of reasoning.

The commonly agreed principles of reasoning are associated with a verification procedure that distinguishes proper applications of these principles from improper ones. A *line of reasoning* is considered valid with respect to such fixed principles (and is thus deemed a proof) if and only if it proceeds by a proper applications of these principles. Thus, a line of reasoning is considered valid if and only if it is accepted by the corresponding verification procedure. This means that, technically speaking, proofs are defined in terms of a predetermined verification procedure (or are define with respect to such a procedure) . Indeed, this state of affairs is best illustrated in the formal study of proofs (i.e., *logic*), which is actually the study of formally defined proof systems: The point is that these proof systems are defined (often explicitly and sometimes only implicitly) in terms of their verification procedures.

The notion of a verification procedure presumes the notion of computation. This fact explains the historical interest of logicians in computer science (cf. [224, 54]). Furthermore, the verification of proofs is supposed to be relatively easy, and hence a natural connection emerges between verification procedures and the notion of efficient computation. This connection was made explicit by complexity theorists, and is captured by the definition of $\mathcal{NP}$ and NP-proof systems (cf. Definition 2.5), which targets all efficient verification procedures.[1]

Recall that Definition 2.5 identifies efficient (verification) procedures with deterministic polynomial-time algorithms, and that it explicitly restricts the length of proofs to be polynomial in the length of the assertion. Thus, *verification is performed in a number of steps that is polynomial in the length of the assertion.* We comment that deterministic proof systems that allow for longer proofs (but require that verification is efficient in terms of the length of the alleged proof) can be modeled as NP-proof systems by adequate padding (of the assertion).

Indeed, NP-proofs provide the ultimate formulation of efficiently verifiable proofs (i.e., proof systems with efficient verification procedures), provided that one associates efficient procedures with *deterministic* polynomial-time algorithms. However, as we shall see, we can gain a lot if we are willing to take a somewhat non-traditional step and allow *probabilistic* (polynomial-time) algorithms and, in particular, *probabilistic* verification procedures.

- Randomized and interactive verification procedures seem much more powerful than their deterministic counterparts.

- Such interactive proof systems allow for the construction of (meaningful) zero-knowledge proofs, which are of great conceptual and practical interest.

- NP-proofs can be efficiently transformed into a (redundant) form that supports super-fast probabilistic verification via very few random probes into the alleged proof.

[1] In contrast, traditional proof systems are formulated based on rules of inference that seem natural in the relevant context. The fact that these inference rules yield an efficient verification procedure is merely a consequence of the correspondence between processes that seem natural and efficient computation.

In all these cases, explicit bounds are imposed on the computational complexity of the verification procedure, which in turn is personified by the notion of a verifier. Furthermore, in all these proof systems, the verifier is allowed to toss coins and rule by statistical evidence. Thus, all these proof systems carry a probability of error; yet, this probability is explicitly bounded and, furthermore, can be reduced by successive application of the proof system.

One important convention. When presenting a proof system, we state all complexity bounds in terms of the length of the assertion to be proved (which is viewed as an input to the verifier). Namely, when we say "polynomial-time" we mean time that is polynomial in the length of this assertion. Indeed, as will become evident, this is *the* natural choice in all the cases that we consider. Note that this convention is consistent with the foregoing discussion of NP-proof systems.[2]

Notational Conventions. We denote by `poly` the set of all integer functions that are upper-bounded by a polynomial, and by `log` the set of all integer functions bounded by a logarithmic function (i.e., $f \in \texttt{log}$ if and only if $f(n) = O(\log n)$). All complexity measures mentioned in this chapter are assumed to be constructible in polynomial-time.

Organization. In Section 9.1 we present the basic definitions and results regarding interactive proof systems. The definition of an interactive proof systems is the starting point for a discussion of zero-knowledge proofs, which is provided in Section 9.2. Section 9.3, which presents the basic definitions and results regarding probabilistically checkable proofs (PCP), can be read independently of the other sections.

Prerequisites: We assume a basic familiarity with elementary probability theory (see Appendix D.1) and randomized algorithms (see Section 6.1).

9.1 Interactive Proof Systems

In light of the growing acceptability of randomized and interactive computations, it is only natural to associate the notion of efficient computation with probabilistic and interactive polynomial-time computations. This leads naturally to the notion of an interactive proof system in which the verification procedure is interactive and randomized, rather than being non-interactive and deterministic. Thus, a "proof" in this context is not a fixed and static object, but rather a randomized (dynamic) process in which the verifier interacts with the prover. Intuitively, one may think of this interaction as consisting of questions asked by the verifier, to which the prover has to reply convincingly.

[2]Recall that Definition 2.5 refers to polynomial-time verification of alleged proofs, which in turn must have length that is bounded by a polynomial in the length of the assertion.

The foregoing discussion, as well as the definition provided in Section 9.1.2, makes explicit reference to a prover, whereas a prover is only implicit in the traditional definitions of proof systems (e.g., NP-proof systems). Before turning to the actual definition, we highlight and further discuss this issue as well as some other conceptual issues.

9.1.1 Motivation and Perspective

We shall discuss the various interpretations given to the notion of a proof in different human contexts, and the attitudes that underly and/or accompany these interpretations. This discussion is aimed at emphasizing that the motivation for the definition of interactive proof systems is not replacing the notion of a mathematical proof, but rather capturing other forms of proofs that are of natural interest. We also discuss the roles of the prover and the verifier, in these settings, and the general notions of completeness and soundness.

9.1.1.1 A static object versus an interactive process

Traditionally in mathematics, a "proof" is a *fixed* sequence consisting of statements that are either self-evident or are derived from previous statements via self-evident rules. Actually, both conceptually and technically, it is more accurate to substitute the phrase "self-evident" by the phrase "commonly agreed" (because, at the last account, self-evidence is a matter of common agreement). In fact, in the formal study of proofs (i.e., logic), the commonly agreed statements are called *axioms*, whereas the commonly agreed rules are referred to as *derivation rules.* We highlight *a key property of mathematics proofs: these proofs are viewed as fixed* (*static*) *objects.*

In contrast, in other areas of human activity, the notion of a "proof" has a much wider interpretation. In particular, a proof is not a fixed object but rather a process by which the validity of an assertion is established. For example, in the context of Law, standing a cross-examination by an opponent, who may ask tough and/or tricky questions, is considered a proof of the facts claimed by the witness. Likewise, various debates that take place in daily life have an analogous potential of establishing claims and are then perceived as proofs. This perception is quite common in philosophical and political debates, and applies even in scientific debates. Needless to say, *a key property of such debates is their interactive* ("dynamic") *nature.* Interestingly, the appealing nature of such "interactive proofs" is reflected in the fact that they are mimicked (in a rigorous manner) in some mathematical *proofs by contradiction*, which emulate an imaginary debate with a potential (generic) skeptic.

Another difference between mathematical proofs and various forms of "daily proofs" is that, while the former aim at certainty, the latter are intended ("only") for establishing claims *beyond any reasonable doubt.* Arguably, an explicitly bounded error probability (as present in our definition of interactive proof systems) is an *extremely strong* form of establishing a claim beyond any reasonable doubt.

We also note that, in mathematics, proofs are often considered more important than their consequence (i.e., the theorem). In contrast, in many daily situations, proofs are considered secondary (in importance) to their consequence. These conflicting attitudes are well-coupled with the difference between written proofs and "interactive" proofs: If one values the proof itself then one may insist on having it archived, whereas if one only cares about the consequence then the way in which it is reached is immaterial.

Interestingly, the foregoing set of daily attitudes (rather than the mathematical ones) will be adequate in the current chapter, where *proofs are viewed merely as a vehicle for the verification of the validity of claims.* (This attitude gets to an extreme in the case of zero-knowledge proofs, where we actually require that the proofs themselves be useless beyond being convincing of the validity of the claimed assertion.)

In general, we will be interested in modeling various forms of proofs that may occur in the world, focusing on proofs that can be verified by automated procedures. These verification procedures are designed to check the validity of potential proofs, and are oblivious of additional features that may appeal to humans such as beauty, insightfulness, etc. In the current section we will consider the most general form of proof systems that still allow efficient verification.

We note that the proof systems that we study refer to mundane theorems (e.g., asserting that a *specific* propositional formula is not satisfiable or that a party sent a message as instructed by a predetermined protocol). We stress that the (meta) theorems that we shall state regarding these proof systems will be proved in the traditional mathematical sense.

9.1.1.2 Prover and Verifier

The wide interpretation of the notion of a proof system, which includes interactive processes of verification, calls for the explicit introduction of two interactive players, called the *prover* and the *verifier*. The verifier is the party that employs the verification procedure, which underlies the definition of any proof system, while the prover is the party that tries to convince the verifier. In the context of static (or non-interactive) proofs, the prover is the transcendental entity providing the proof, and thus in this context the prover is often not mentioned at all (when discussing the verification of alleged proofs). Still, explicitly mentioning potential provers may be beneficial even when discussing such static (non-interactive) proofs.

We highlight the "distrustful attitude" towards the prover, which underlies any proof system. If the verifier trusts the prover then no proof is needed. Hence, whenever discussing a proof system, one should envision a setting in which the verifier is not trusting the prover, and furthermore is skeptic of anything that the prover says. In such a setting the prover's goal is to convince the verifier, while the verifier should make sure that it is not fooled by the prover. (See further discussion in §9.1.1.3.) Note that the verifier is "trusted" to protect its own interests by employing the predetermined verification procedure; indeed, the asymmetry with respect to who we trust is an artifact of our focus on the verification process (or task). In general, each party is trusted to protect its own interests (i.e., the verifier is trusted to protect its own interests), but no party is trusted to protect the interests of the other party (i.e., the prover is not trusted to protect the verifier's interest of not being fooled by the prover).

Another asymmetry between the two parties is that our discussion focuses on the complexity of the verification task and ignores (as a first approximation) the complexity of the proving task (which is only discussed in §9.1.5.1). Note that this asymmetry is reflected in the definition of NP-proof systems; that is, verification is required to be efficient, whereas for sets $\mathcal{NP} \setminus \mathcal{P}$ finding adequate proofs is infeasible. Thus, as a first approximation, we consider the question of what can be efficiently verified when interacting with an arbitrary prover (which may be infinitely powerful). Once this question is resolved, we shall also consider the complexity of the proving task (indeed, see §9.1.5.1).

9.1.1.3 Completeness and Soundness

Two fundamental properties of a proof system (i.e., of a verification procedure) are its *soundness* (or *validity*) and *completeness*. The soundness property asserts that the verification procedure cannot be "tricked" into accepting false statements. In other words, *soundness* captures the verifier's ability to protect itself from being convinced of false statements (no matter what the prover does in order to fool it). On the other hand, *completeness* captures the ability of some prover to convince the verifier of true statements (belonging to some predetermined set of true statements). Note that both properties are essential to the very notion of a proof system.

We note that not every set of true statements has a "reasonable" proof system in which each of these statements can be proved (while no false statement can be "proved"). This fundamental phenomenon is given a precise meaning in results such as *Gödel's Incompleteness Theorem* and Turing's theorem regarding the *undecidability of the Halting Problem*. In contrast, recall that $\mathcal{NP}$ was defined as the class of sets having proof systems that support efficient deterministic verification (of "written proofs"). This section is devoted to the study of a more liberal notion of efficient verification procedures (allowing both randomization and interaction).

9.1.2 Definition

Loosely speaking, an interactive proof is a game between a computationally bounded verifier and a computationally unbounded prover whose goal is to convince the verifier of the validity of some assertion. Specifically, the verifier employs a probabilistic polynomial-time strategy (whereas no computational restrictions apply to the prover's strategy). It is required that if the assertion holds then the verifier always accepts (i.e., when interacting with an appropriate prover strategy). On the other hand, if the assertion is false then the verifier must reject with probability at least $\frac{1}{2}$, no matter what strategy is being employed by the prover. (The error probability can be reduced by running such a proof system several times.)

We formalize the interaction between parties by referring to the *strategies* that the parties employ.[3] A strategy for a party describes the *party's next move* (i.e., its next message or its final decision) *as a function of the common input* (i.e., the aforementioned assertion), *its internal coin tosses, and all messages it has received so far*. That is, we assume that each party records the outcomes of its past coin tosses as well as all the messages it has received, and determines its moves based on these. Thus, an interaction between two parties, employing strategies A and B respectively, is determined by the common input, denoted x, and the randomness of both parties, denoted r_A and r_B. Assuming that A takes the first move (and B takes the last move), the corresponding (t-round) interaction transcript (on common input x and randomness r_A and r_B) is $\alpha_1, \beta_1, ..., \alpha_t, \beta_t$, where $\alpha_i = A(x, r_A, \beta_1, ..., \beta_{i-1})$ and $\beta_i = B(x, r_B, \alpha_1, ..., \alpha_i)$. The corresponding final decision of A is defined as $A(x, r_A, \beta_1, ..., \beta_t)$.

We say that a party employs a probabilistic polynomial-time strategy if its next move can be computed in a number of steps that is *polynomial in the length of the common input*. In particular, this means that, on input common input x, the strategy may only consider a polynomial in $|x|$ many messages, which are each of poly$(|x|)$ length.[4] Intuitively, if the other party exceeds an a priori (polynomial in $|x|$) bound on the total length of the messages that it is allowed to send, then the execution is suspended. Thus, referring to the aforementioned strategies, we say that A is a probabilistic polynomial-time strategy if, for every i and $r_A, \beta_1, ..., \beta_i$, the value of $A(x, r_A, \beta_1, ..., \beta_i)$ can be computed in time polynomial in $|x|$. Again, in proper use, it must hold that $|r_A|, t$ and the $|\beta_i|$'s are all polynomial in $|x|$.

Definition 9.1 (Interactive Proof systems – IP):[5] *An* interactive proof system for a set S *is a two-party game, between a* verifier *executing a* probabilistic polynomial-time strategy*, denoted* V*, and a* prover *that executes a* (computationally unbounded) *strategy, denoted* P*, satisfying the following two conditions:*

- Completeness: *For every $x \in S$, the verifier V always accepts after interacting with the prover P on common input x.*
- Soundness: *For every $x \notin S$ and every strategy P^*, the verifier V rejects with probability at least $\frac{1}{2}$ after interacting with P^* on common input x.*

We denote by $\mathcal{IP}$ the class of sets having interactive proof systems.

The error probability (in the soundness condition) can be reduced by successive applications of the proof system. (This is easy to see in the case of sequential repetitions, but holds also for parallel repetitions; see Exercise 9.1.) In particular,

[3] An alternative formulation refers to the interactive machines that capture the behavior of each of the parties (see, e.g., [90, Sec. 4.2.1.1]). Such an interactive machine invokes the corresponding strategy, while handling the communication with the other party and keeping a record of all messages received so far.

[4] Needless to say, the number of internal coin tosses fed to a polynomial-time strategy must also be bounded by a polynomial in the length of x.

[5] We follow the convention of specifying strategies for both the verifier and the prover. An alternative presentation only specifies the verifier's strategy, while rephrasing the completeness condition as follows: *There exists a prover strategy P such that, for every $x \in S$, the verifier V always accepts after interacting with P on common input x.*

repeating the proving process for k times, reduces the probability that the verifier is fooled (i.e., accepts a false assertion) to 2^{-k}, and we can afford doing so for any $k = \text{poly}(|x|)$. Variants on the basic definition are discussed in Section 9.1.4.

The role of randomness. Randomness is essential to the power of interactive proofs; that is, restricting the verifier to deterministic strategies yields a class of interactive proof systems that has no advantage over the class of NP-proof systems. The reason being that, in case the verifier is deterministic, the prover can predict the verifier's part of the interaction. Thus, the prover can just supply its own sequence of answers to the verifier's sequence of (predictable) questions, and the verifier can just check that these answers are convincing. Actually, we establish that soundness error (and not merely randomized verification) is essential to the power of interactive proof systems (i.e., their ability to reach beyond NP-proofs).

Proposition 9.2 *Suppose that S has an interactive proof system (P, V) with* no soundness error*; that is, for every $x \notin S$ and every potential strategy P^*, the verifier V rejects with probability* one *after interacting with P^* on common input x. Then $S \in \mathcal{NP}$.*

Proof: We may assume, without loss of generality, that V is deterministic (by just fixing arbitrarily the contents of its random-tape (e.g., to the all-zero string) and noting that both (perfect) completeness and perfect (i.e., errorless) soundness still hold). Thus, the case of zero soundness error reduces to the case of deterministic verifiers.

Now, since V is deterministic, the prover can predict each message sent by V, because each such message is uniquely determined by the common input and the previous prover messages. Thus, a sequence of optimal prover's messages (i.e., a sequence of messages leading V to accept $x \in S$) can be (pre)determined (without interacting with V) *based solely on the common input x*.[6] Hence, $x \in S$ if and only if there exists a sequence of (prover's) messages that make (the deterministic) V accept x, where the question of whether a specific sequence (of prover's messages) makes V accept x depends only on the sequence and on the common input x (because V tosses no coins that may affect this decision).[7] The foregoing condition can be checked in polynomial-time, and so a "passing sequence" constitutes an NP-witness for $x \in S$. It follows that $S \in \mathcal{NP}$. ■

Reflection. The moral of the reasoning underlying the proof Proposition 9.2 is that *there is no point to interact with a party whose moves are easily predictable*, because such moves can be determined without any interaction. This moral represents the prover's point of view (regarding interaction with deterministic verifiers).

[6] As usual, we do not care about the complexity of determining such a sequence, since no computational bounds are placed on the prover.

[7] Recall that in the case that V is randomized, its final decision also depends on its internal coin tosses (and not only on the common input and on the sequence of prover's messages). In that case, the verifier's own messages may reveal information about the verifier's internal coin tosses, which in turn may help the prover to answer with convincing messages.

In contrast, even an infinitely powerful party (e.g., a prover) may gain by interacting with an unpredictable party (e.g., a randomized verifier), because this interaction may provide useful information (e.g., information regarding the verifier's coin tosses, which in turn allows the prover to increase its probability of answering convincingly). Furthermore, from the verifier's point of view it is beneficial to interact with the prover, because the latter is computationally stronger (and thus its moves may not be *easily* predictable by the verifier even in the case that they are predictable in an information theoretic sense).

9.1.3 The Power of Interactive Proofs

We have seen that randomness is essential to the power of interactive proof systems in the sense that without randomness interactive proofs are not more powerful than NP-proofs. Indeed, the power of interactive proof arises from the combination of randomization and interaction. We first demonstrate this point by a simple proof system for a specific coNP-set that is not known to have an NP-proof system, and next prove the celebrated result $\mathcal{IP} = \mathcal{PSPACE}$, which suggests that interactive proofs are much stronger than NP-proofs.

9.1.3.1 A simple example

> *One day on the Olympus, bright-eyed Athena claimed that Nectar poured out of the new silver-coated jars tastes less good than Nectar poured out of the older gold-decorated jars. Mighty Zeus, who was forced to introduce the new jars by the practically oriented Hera, was annoyed at the claim. He ordered that Athena be served one hundred glasses of Nectar, each poured at random either from an old jar or from a new one, and that she tell the source of the drink in each glass. To everybody's surprise, wise Athena correctly identified the source of each serving, to which the Father of the Gods responded "my child, you are either right or extremely lucky." Since all gods knew that being lucky was not one of the attributes of Pallas-Athena, they all concluded that the impeccable goddess was right in her claim.*

The foregoing story illustrates the main idea underlying the interactive proof for Graph Non-Isomorphism, presented in Construction 9.3. Informally, this interactive proof system is designed for proving dissimilarity of two given objects (in the foregoing story these are the two brands of Nectar, whereas in Construction 9.3 these are two non-isomorphic graphs). We note that, typically, proving similarity between objects is easy, because one can present a mapping (of one object to the other) that demonstrates this similarity. In contrast, proving dissimilarity seems harder, because in general there seems to be no succinct proof of dissimilarity (e.g., clearly, showing that a particular mapping fails does not suffice, while enumerating all possible mappings (and showing that each fails) does not yield a succinct proof). More generally, it is typically easy to prove the existence of an easily verifiable structure in a given object by merely presenting this structure, but proving the non-existence of such a structure seems hard. Formally, membership in an NP-set is proved by presenting an NP-witness, but it is not clear how to prove the non-existence of such a witness. Indeed, recall that the common belief is that co$\mathcal{NP} \neq \mathcal{NP}$.

Two graphs, $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, are called isomorphic if there exists a 1-1 and onto mapping, ϕ, from the vertex set V_1 to the vertex set V_2 such that $\{u, v\} \in E_1$ if and only if $\{\phi(v), \phi(u)\} \in E_2$. This ("edge preserving") mapping ϕ, in case it exists, is called an *isomorphism* between the graphs. The following protocol specifies a way of proving that two graphs are not isomorphic, while it is not known whether such a statement can be proved via a non-interactive process (i.e., via an NP-proof system).

Construction 9.3 (Interactive proof for Graph Non-Isomorphism):

- Common Input: *A pair of graphs,* $G_1 = (V_1, E_1)$ *and* $G_2 = (V_2, E_2)$.
- Verifier's first step (V1): *The verifier selects at random one of the two input graphs, and sends to the prover a random isomorphic copy of this graph. Namely, the verifier selects uniformly* $\sigma \in \{1, 2\}$, *and a random permutation* π *from the set of permutations over the vertex set* V_σ. *The verifier constructs a graph with vertex set* V_σ *and edge set*
$$E \stackrel{\text{def}}{=} \{\{\pi(u), \pi(v)\} : \{u, v\} \in E_\sigma\}$$
and sends (V_σ, E) *to the prover.*
- Motivating Remark: *If the input graphs are non-isomorphic, as the prover claims, then the prover should be able to distinguish (not necessarily by an efficient algorithm) isomorphic copies of one graph from isomorphic copies of the other graph. However, if the input graphs are isomorphic, then a random isomorphic copy of one graph is distributed identically to a random isomorphic copy of the other graph.*
- Prover's step: *Upon receiving a graph,* $G' = (V', E')$, *from the verifier, the prover finds a* $\tau \in \{1, 2\}$ *such that the graph* G' *is isomorphic to the input graph* G_τ. *(If both* $\tau = 1, 2$ *satisfy the condition then* τ *is selected arbitrarily. In case no* $\tau \in \{1, 2\}$ *satisfies the condition,* τ *is set to 0). The prover sends* τ *to the verifier.*
- Verifier's second step (V2): *If the message,* τ, *received from the prover equals* σ *(chosen in Step V1) then the verifier outputs 1 (i.e., accepts the common input). Otherwise the verifier outputs 0 (i.e., rejects the common input).*

The verifier's strategy in Construction 9.3 is easily implemented in probabilistic polynomial-time. We do not known of a probabilistic polynomial-time implementation of the prover's strategy, but this is not required. The motivating remark justifies the claim that Construction 9.3 constitutes an interactive proof system for the set of pairs of non-isomorphic graphs.[8] Recall that the latter is a co$\mathcal{NP}$-set (which is not known to be in $\mathcal{NP}$).

9.1.3.2 The full power of interactive proofs

The interactive proof system of Construction 9.3 refers to a specific coNP-set that is not known to be in $\mathcal{NP}$. It turns out that interactive proof systems are powerful enough to prove membership in *any* coNP-set (e.g., prove that a graph is not 3-colorable). Thus, assuming that $\mathcal{NP} \neq \text{co}\mathcal{NP}$, this establishes that interactive proof systems are more powerful than NP-proof systems. Furthermore, the class of sets having interactive proof systems coincides with the class of sets that can be decided using a polynomial amount of work-space.

Theorem 9.4 (The IP Theorem): $\mathcal{IP} = \mathcal{PSPACE}$.

Recall that it is widely believed that $\mathcal{NP}$ is a *proper* subset of $\mathcal{PSPACE}$. Thus, under this conjecture, interactive proofs are more powerful than NP-proofs.

Sketch of the Proof of Theorem 9.4

We first show that $\text{co}\mathcal{NP} \subseteq \mathcal{IP}$, by presenting an interactive proof system for the co$\mathcal{NP}$-complete set of unsatisfiable CNF formulae. Next we extend this proof system to obtain one for the $\mathcal{PSPACE}$-complete set of unsatisfiable Quantified Boolean Formulae. Finally, we observe that $\mathcal{IP} \subseteq \mathcal{PSPACE}$. Indeed, proving that some co$\mathcal{NP}$-complete set has an interactive proof system is the core of the proof of Theorem 9.4 (see Exercise 9.2).

We show that the set of unsatisfiable CNF formulae has an interactive proof system by using algebraic methods, which are *applied to an arithmetic generalization of the said Boolean problem* (rather than to the problem itself). That is, in order to demonstrate that this Boolean problem has an interactive proof system, we first introduce an arithmetic generalization of CNF formulae, and then construct an interactive proof system for the resulting arithmetic assertion (by capitalizing on the arithmetic formulation of the assertion). Intuitively, we present an iterative process, which involves interaction between the prover and the verifier, such that in each iteration the residual claim to be established becomes simpler (i.e., contains one variable less). This iterative process seems to be enabled by the fact that the various claims refer to the arithmetic problem rather than to the original Boolean problem. (Actually, one may say that the key point is that these claims refer to a generalized problem rather than to the original one.)

[8] In case G_1 is not isomorphic to G_2, no graph can be isomorphic to both input graphs (i.e., both to G_1 and to G_2). In this case the graph G' sent in Step (V1) uniquely determines the bit σ. On the other hand, if G_1 and G_2 are isomorphic then, for every G' sent in Step (V1), the number of isomorphisms between G_1 and G' equals the number of isomorphisms between G_2 and G'. It follows that, in this case G', yields no information about σ (chosen by the verifier), and so no prover may convince the verifier with probability exceeding $1/2$.

Teaching note: We devote most of the presentation to establishing that co$\mathcal{NP} \subseteq \mathcal{IP}$, and recommend doing the same in class. Our presentation focuses on the main ideas, and neglects some minor implementation details (which can be found in [161, 204]).

The starting point: We prove that co$\mathcal{NP} \subseteq \mathcal{IP}$ by presenting an interactive proof system for the set of unsatisfiable CNF formulae, which is co$\mathcal{NP}$-complete. Thus, our starting point is a given Boolean CNF formula, which is claimed to be unsatisfiable.

Arithmetization of Boolean (CNF) formulae: Given a Boolean (CNF) formula, we replace the Boolean variables by integer variables, and replace the logical operations by corresponding arithmetic operations. In particular, the Boolean values `false` and `true` are replaced by the integer values 0 and 1 (respectively), OR-clauses are replaced by sums, and the top level conjunction is replaced by a product. This translation is depicted in Figure 9.1. Note that the Boolean formula

	BOOLEAN	ARITHMETIC
variable values	`false`, `true`	0, 1
connectives	$\neg x$, $\vee$ and $\wedge$	$1 - x$, $+$ and $\cdot$
final values	`false`, `true`	0, positive

Figure 9.1: Arithmetization of CNF formulae.

is satisfied (resp., unsatisfied) by a specific truth assignment if and only if evaluating the resulting arithmetic expression at the corresponding 0-1 assignment yields a positive (integer) value (resp., yields the value zero). Thus, the claim that the original Boolean formula is unsatisfiable translates to the claim that the summation of the resulting arithmetic expression, over all 0-1 assignments to its variables, yields the value zero. For example, the Boolean formula

$$(x_3 \vee \neg x_5 \vee x_{17}) \wedge (x_5 \vee x_9) \wedge (\neg x_3 \vee \neg x_4)$$

is replaces by the arithmetic expression

$$(x_3 + (1 - x_5) + x_{17}) \cdot (x_5 + x_9) \cdot ((1 - x_3) + (1 - x_4))$$

and the Boolean formula is unsatisfiable if and only if the sum of the corresponding arithmetic expression, taken over all choices of $x_1, x_2, ..., x_{17} \in \{0, 1\}$, equals 0. Thus, *proving that the original Boolean formula is unsatisfiable reduces to proving that the corresponding arithmetic summation evaluates to 0.* We highlight two additional observations regarding the resulting arithmetic expression:

1. The arithmetic expression is a low degree polynomial over the integers; specifically, its (total) degree equals the number of clauses in the original Boolean formula.

2. For any Boolean formula, the value of the corresponding arithmetic expression (for any choice of $x_1, ..., x_n \in \{0,1\}$) resides within the interval $[0, v^m]$, where v is the maximum number of variables in a clause, and m is the number of clauses. Thus, summing over all 2^n possible 0-1 assignments, where $n \leq vm$ is the number of variables, yields an integer value in $[0, 2^n v^m]$.

Moving to a Finite Field: In general, whenever we need to check equality between two integers in $[0, M]$, it suffices to check their equality mod q, where $q > M$. The benefit is that, if q is prime then the arithmetic is now in a finite field (mod q), and so certain things are "nicer" (e.g., uniformly selecting a value). Thus, proving that a CNF formula is not satisfiable reduces to proving an equality of the following form

$$\sum_{x_1=0,1} \cdots \sum_{x_n=0,1} \phi(x_1, ..., x_n) \equiv 0 \pmod{q}, \tag{9.1}$$

where ϕ is a low-degree multi-variate polynomial (and q can be represented using $O(|\phi|)$ bits). In the rest of this exposition, all arithmetic operations refer to the finite field of q elements, denoted $\mathrm{GF}(q)$.

Overview of the actual protocol: stripping summations in iterations. Given a formal expression as in Eq. (9.1), we strip off summations in iterations, stripping a single summation at each iteration, and instantiate the corresponding free variable as follows. At the beginning of each iteration the prover is supposed to supply the univariate polynomial representing the residual expression as a function of the (single) currently stripped variable. (By Observation 1, this is a low degree polynomial and so it has a short description.)[9] The verifier checks that the polynomial (say, p) is of low degree, and that it corresponds to the current value (say, v) being claimed (i.e., it verifies that $p(0) + p(1) \equiv v$). Next, the verifier randomly instantiates the currently free variable (i.e., it selects uniformly $r \in \mathrm{GF}(q)$), yielding a new value to be claimed for the resulting expression (i.e., the verifier computes $v \leftarrow p(r)$, and expects a proof that the residual expression equals v). The verifier sends the uniformly chosen instantiation (i.e., r) to the prover, and the parties proceed to the next iteration (which refers to the residual expression and to the new value v). At the end of the last iteration, the verifier has a closed form expression (i.e., an expression without formal summations), which can be easily checked against the claimed value.

A single iteration (detailed): The i^{th} iteration is aimed at proving a claim of the form

$$\sum_{x_i=0,1} \cdots \sum_{x_n=0,1} \phi(r_1, ..., r_{i-1}, x_i, x_{i+1}, ..., x_n) \equiv v_{i-1} \pmod{q}, \tag{9.2}$$

[9] We also use Observation 2, which implies that we may use a finite field with elements having a description length that is polynomial in the length of the original Boolean formula (i.e., $\log_2 q = O(vm)$).

where $v_0 = 0$, and $r_1, ..., r_{i-1}$ and v_{i-1} are as determined in previous iterations. The i^{th} iteration consists of two steps (messages): a prover step followed by a verifier step. The prover is supposed to provide the verifier with the univariate polynomial p_i that satisfies

$$p_i(z) \stackrel{\text{def}}{=} \sum_{x_{i+1}=0,1} \cdots \sum_{x_n=0,1} \phi(r_1, ..., r_{i-1}, z, x_{i+1}, ..., x_n) \bmod q\,. \tag{9.3}$$

Note that, module q, the value $p_i(0)+p_i(1)$ equals the l.h.s of Eq. (9.2). Denote by p'_i the actual polynomial sent by the prover (i.e., the honest prover sets $p'_i = p_i$). Then, the verifier first checks if $p'_i(0)+p'_i(1) \equiv v_{i-1} \pmod q$, and next uniformly selects $r_i \in \mathrm{GF}(q)$ and sends it to the prover. Needless to say, the verifier will reject if the first check is violated. The claim to be proved in the next iteration is

$$\sum_{x_{i+1}=0,1} \cdots \sum_{x_n=0,1} \phi(r_1, ..., r_{i-1}, r_i, x_{i+1}, ..., x_n) \equiv v_i \pmod q, \tag{9.4}$$

where $v_i \stackrel{\text{def}}{=} p'_i(r_i) \bmod q$ is computed by each party.

Completeness of the protocol: When the initial claim (i.e., Eq. (9.1)) holds, the prover can supply the correct polynomials (as determined in Eq. (9.3)), and this will lead the verifier to always accept.

Soundness of the protocol: It suffices to upper-bound the probability that, for a particular iteration, the entry claim (i.e., Eq. (9.2)) is false while the ending claim (i.e., Eq. (9.4)) is valid. Indeed, let us focus on the i^{th} iteration, and let v_{i-1} and p_i be as in Eq. (9.2) and Eq. (9.3), respectively; that is, v_{i-1} is the (wrong) value claimed at the beginning of the i^{th} iteration and p_i is the polynomial representing the expression obtained when stripping the current variable (as in Eq. (9.3)). Let $p'_i(\cdot)$ be any potential answer by the prover. We may assume, without loss of generality, that $p'_i(0) + p'_i(1) \equiv v_{i-1} \pmod q$ and that p'_i is of low-degree (since otherwise the verifier will definitely reject). Using our hypothesis (that the entry claim of Eq. (9.2) is false), we know that $p_i(0) + p_i(1) \not\equiv v_{i-1} \pmod q$. Thus, p'_i and p_i are different low-degree polynomials, and so they may agree on very few points (if at all). Now, if the verifier's instantiation (i.e., its choice of a random r_i) does not happen to be one of these few points (i.e., $p_i(r_i) \not\equiv p'_i(r_i) \pmod q$), then the ending claim (i.e., Eq. (9.4)) is false too (because the new value (i.e., v_i) is set to $p'_i(r_i) \bmod q$, while the residual expression evaluates to $p_i(r_i)$). Details are left as an exercise (see Exercise 9.3).

This establishes that the set of unsatisfiable CNF formulae has an interactive proof system. Actually, a similar proof system (which uses a related arithmetization – see Exercise 9.5) can be used to prove that a given formula has a given number of satisfying assignment; i.e., prove membership in the ("counting") set

$$\{(\phi, k) : |\{\tau : \phi(\tau) = 1\}| = k\}\,. \tag{9.5}$$

Using adequate reductions, it follows that every problem in $\#\mathcal{P}$ has an interactive proof system (i.e., for every $R \in \mathcal{PC}$, the set $\{(x,k) : |\{y : (x,y) \in R\}| = k\}$ is in $\mathcal{IP}$). Proving that $\mathcal{PSPACE} \subseteq \mathcal{IP}$ requires a little more work, as outlined next.

Obtaining interactive proofs for PSPACE (the basic idea). We present an interactive proof for the set of satisfied Quantified Boolean Formulae (QBF), which is complete for $\mathcal{PSPACE}$ (see Theorem 5.15).[10] Recall that the number of quantifiers in such formulae is unbounded (e.g., it may be polynomially related to the length of the input), that there are both existential and universal quantifiers, and furthermore these quantifiers may alternate. In the arithmetization of these formulae, we replace existential quantifiers by summations and universal quantifiers by products. Two difficulties arise when considering the application of the foregoing protocol to the resulting arithmetic expression. Firstly, the (integral) value of the expression (which may involve a big number of nested formal products) is only upper-bounded by a double-exponential function (in the length of the input). Secondly, when stripping a summation (or a product), the expression may be a polynomial of high degree (due to nested formal products that may appear in the remaining expression).[11] For example, both phenomena occur in the following expression

$$\sum_{x=0,1} \prod_{y_1=0,1} \cdots \prod_{y_n=0,1} (x+y_n),$$

which equals $\sum_{x=0,1} x^{2^{n-1}} \cdot (1+x)^{2^{n-1}}$. The first difficulty is easy to resolve by using the fact (to be established in Exercise 9.7) that if two integers in $[0,M]$ are different then they must be different modulo most of the primes in the interval $[3,\text{poly}(\log M)]$. Thus, we let the verifier selects a random prime q of length that is linear in the length of the original formula, and the two parties consider the arithmetic expression reduced modulo this q. The second difficulty is resolved by noting that $\mathcal{PSPACE}$ is actually reducible to a special form of (non-canonical) QBF in which no variable appears both to the left and to the right of more than one universal quantifier (see the proof of Theorem 5.15 or alternatively Exercise 9.6). It follows that when arithmetizing and stripping summations (or products) from the resulting arithmetic expression, the corresponding univariate polynomial is of low degree (i.e., at most twice the length of the original formula, where the factor

[10] Actually, the following extension of the foregoing proof system yields a proof system for the set of *unsatisfied* Quantified Boolean Formulae (which is also complete for $\mathcal{PSPACE}$). Alternatively, an interactive proof system for QBF can be obtained by extending the related proof system presented in Exercise 9.5.

[11] This high degree causes two difficulties, where only the second one is acute. The first difficulty is that the soundness of the corresponding protocol will require working in a finite field that is sufficiently larger than this high degree, but we can afford doing so (since the degree is at most exponential in the formula's length). The second (and more acute) difficulty is that the polynomial may have a large (i.e., exponential) number of non-zero coefficients and so the verifier cannot afford to read the standard representation of this polynomial (as a list of all non-zero coefficients). Indeed, other succinct and effective representations of such polynomials may exist in some cases (as in the following example), but it is unclear how to obtain such representations in general.

of two is due to the single universal quantifier that has this variable quantified on its left and appearing on its right).

IP is contained in PSPACE: We shall show that, for every interactive proof system, there exists an *optimal prover strategy* that can be implemented in polynomial-space, where an optimal prover strategy is one that maximizes the probability that the prescribed verifier accepts the common input. It follows that $\mathcal{IP} \subseteq \mathcal{PSPACE}$, because (for every $S \in \mathcal{IP}$) we can emulate, in polynomial space, all possible interactions of the prescribed verifier with any fixed polynomial-space prover strategy (e.g., an optimal one).

Proposition 9.5 *Let V be a probabilistic polynomial-time* (verifier) *strategy. Then, there exists a polynomial-space computable* (prover) *strategy f that, for every x, maximizes the probability that V accepts x. That is, for every P^* and every x it holds that the probability that V accepts x after interacting with P^* is upper-bounded by the probability that V accepts x after interacting with f.*

Proof Sketch: For every common input x and any possible partial transcript γ of the interaction so far, the strategy[12] f determines an optimal next-message for the prover by considering all possible coin tosses of the verifier that are consistent with (x, γ). Specifically, f is determined *recursively* such that $f(x, \gamma) = m$ if m maximizes the number of outcomes of the verifier's coin-tosses that are consistent with (x, γ) and lead the verifier to accept when subsequent prover moves are determined by f (which is where recursion is used). That is, the verifier's random sequence r support the setting $f(x, \gamma) = m$, where $\gamma = (\alpha_1, \beta_1, ..., \alpha_t, \beta_t)$, if the following two conditions hold:

1. r is consistent with (x, γ), which means that for every $i \in \{1, ..., t\}$ it holds that $\beta_i = V(x, r, \alpha_1, ..., \alpha_i)$.
2. r leads V to accept when the subsequent prover moves are determined by f, which means at termination (i.e., after T rounds) it holds that

$$V(x, r, \alpha_1, ..., \alpha_t, m, \alpha_{t+2}, ..., \alpha_T) = 1\,,$$

   where for every $i \in \{t+1, ..., T-1\}$ it holds that $\alpha_{i+1} = f(x, \gamma, m, \beta_{t+1}, ..., \alpha_i, \beta_i)$ and $\beta_i = V(x, r, \alpha_1, ..., \alpha_t, m, \alpha_{t+2}, ..., \alpha_i)$.

Thus, $f(x, \gamma) = m$ if m maximizes the value of $\mathsf{E}[\xi_{f,V}(x, R_\gamma, \gamma, m)]$, where R_γ is selected uniformly among the r's that are consistent with (x, γ) and $\xi_{f,V}(x, r, \gamma, m)$ indicates whether or not V accepts x in the subsequent interaction with f (which refers to randomness r and partial transcript (γ, m)). It follows that the value $f(x, \gamma)$ can be computed in polynomial-space when given oracle access to $f(x, \gamma, \cdot, \cdot)$. The proposition follows by standard composition of space-bounded computations (i.e., allocating separate space to each level of the recursion, while using the same space in all recursive calls of each level). □

[12] For sake of convenience, when describing the strategy f, we refer to the entire partial transcript of the interaction with V (rather than merely to the sequence of previous messages sent by V).

9.1.4 Variants and finer structure: an overview

In this subsection we consider several variants on the basic definition of interactive proofs as well as finer complexity measures. This is an advanced subsection, which only provides an overview of the various notions and results (as well as pointers to proofs of the latter).

9.1.4.1 Arthur-Merlin games a.k.a public-coin proof systems

The verifier's messages in a general interactive proof system are determined arbitrarily (but efficiently) based on the verifier's view of the interaction so far (which includes its internal coin tosses, which without loss of generality can take place at the onset of the interaction). Thus, the verifier's past coin tosses are not necessarily revealed by the messages that it sends. In contrast, in public-coin proof systems (a.k.a Arthur-Merlin proof systems), the verifier's messages contain the outcome of any coin that it tosses *at the current round*. Thus, these messages reveal the randomness used towards generating them (i.e., this randomness becomes public). Actually, without loss of generality, the verifier's messages can be identical to the outcome of the coins tossed at the current round (because any other string that the verifier may compute based on these coin tosses is actually determined by them).

Note that the proof systems presented in the proof of Theorem 9.4 are of the public-coin type, whereas this is not the case for the Graph Non-Isomorphism proof system (of Construction 9.3). Thus, although not all natural proof systems are of the public-coin type, by Theorem 9.4 every set having an interactive proof system also has a public-coin interactive proof system. This means that, *in the context of interactive proof systems, asking random questions is as powerful as asking clever questions.* (A stronger statement appears at the end of §9.1.4.3.)

Indeed, public-coin proof systems are a syntactically restricted type of interactive proof systems. This restriction may make the design of such systems more difficult, but potentially facilitates their analysis (and especially when the analysis refers to a generic system). Another advantage of public-coin proof systems is that the verifier's actions (except for its final decision) are oblivious of the prover's messages. This property is used in the proof of Theorem 9.12.

9.1.4.2 Interactive proof systems with two-sided error

In Definition 9.1 error probability is allowed in the soundness condition but not in the completeness condition. In such a case, we say that the proof system has perfect completeness (or one-sided error probability). A more general definition allows an error probability (upper-bounded by, say, 1/3) in both the completeness and the soundness conditions. Note that sets having such generalized (two-sided error) interactive proofs are also in $\mathcal{PSPACE}$, and thus (by Theorem 9.4) allowing two-sided error does not increase the power of interactive proofs. See further discussion at the end of §9.1.4.3.

9.1.4.3 A hierarchy of interactive proof systems

Definition 9.1 only refers to the *total* computation time of the verifier, and thus allows an arbitrary (polynomial) number of messages to be exchanged. A finer definition refers to the number of messages being exchanged (also called the number of rounds).[13]

Definition 9.6 (The round-complexity of interactive proof):

- *For an integer function m, the complexity class $\mathcal{IP}(m)$ consists of sets having an interactive proof system in which, on common input x, at most $m(|x|)$ messages are exchanged between the parties.*[14]
- *For a set of integer functions, M, we let $\mathcal{IP}(M) \stackrel{\text{def}}{=} \bigcup_{m \in M} \mathcal{IP}(m)$. Thus, $\mathcal{IP} = \mathcal{IP}(\mathtt{poly})$.*

For example, interactive proof systems in which the verifier sends a single message that is answered by a single message of the prover corresponds to $\mathcal{IP}(2)$. Clearly, $\mathcal{NP} \subseteq \mathcal{IP}(1)$, yet the inclusion may be strict because in $\mathcal{IP}(1)$ the verifier may toss coins after receiving the prover's single message. (Also note that $\mathcal{IP}(0) = \text{co}\mathcal{RP}$.)

Definition 9.6 gives rise to a natural hierarchy of interactive proof systems, where different "levels" of this hierarchy correspond to different "growth rates" of the round-complexity of these systems. The following results are known regarding this hierarchy.

- A linear speed-up (see Appendix F.2 (or [22] and [110])): For every integer function, f, such that $f(n) \geq 2$ for all n, the class $\mathcal{IP}(O(f(\cdot)))$ collapses to the class $\mathcal{IP}(f(\cdot))$. In particular, $\mathcal{IP}(O(1))$ collapses to $\mathcal{IP}(2)$.
- The class $\mathcal{IP}(2)$ contains sets that are not known to be in $\mathcal{NP}$; e.g., Graph Non-Isomorphism (see Construction 9.3). However, under plausible intractability assumptions, $\mathcal{IP}(2) = \mathcal{NP}$ (see [166]).
- If $\text{co}\mathcal{NP} \subseteq \mathcal{IP}(2)$ then the Polynomial-Time Hierarchy collapses (see [44]).

It is conjectured that $\text{co}\mathcal{NP}$ is *not* contained in $\mathcal{IP}(2)$, and consequently that interactive proofs with an unbounded number of message exchanges are more powerful than interactive proofs in which only a bounded (i.e., constant) number of messages are exchanged.[15]

The class $\mathcal{IP}(1)$, also denoted $\mathcal{MA}$, seems to be *the* "real" randomized (and yet non-interactive) version of $\mathcal{NP}$: Here the prover supplies a candidate (polynomial-size) "proof", and the verifier assesses its validity probabilistically (rather than deterministically).

[13] An even finer structure emerges when considering also the total length of the messages sent by the prover (see [105]).

[14] We count the total number of messages exchanged regardless of the direction of communication.

[15] Note that the linear speed-up cannot be applied for an unbounded number of times, because each application may increase (e.g., square) the time-complexity of verification.

The IP-hierarchy (i.e., $\mathcal{IP}(\cdot)$) equals an analogous hierarchy, denoted $\mathcal{AM}(\cdot)$, that refers to public-coin (a.k.a Arthur-Merlin) interactive proofs. That is, for every integer function f, it holds that $\mathcal{AM}(f) = \mathcal{IP}(f)$. For $f \geq 2$, it is also the case that $\mathcal{AM}(f) = \mathcal{AM}(O(f))$; actually, the aforementioned linear speed-up for $\mathcal{IP}(\cdot)$ is established by combining the following two results:

1. Emulating $\mathcal{IP}(\cdot)$ by $\mathcal{AM}(\cdot)$ (see §F.2.1 or [110]): $\mathcal{IP}(f) \subseteq \mathcal{AM}(f+3)$.

2. Linear speed-up for $\mathcal{AM}(\cdot)$ (see §F.2.2 or [22]): $\mathcal{AM}(2f) \subseteq \mathcal{AM}(f+1)$.

In particular, $\mathcal{IP}(O(1)) = \mathcal{AM}(2)$, even if $\mathcal{AM}(2)$ is restricted such that the verifier tosses no coins after receiving the prover's message. (Note that $\mathcal{IP}(1) = \mathcal{AM}(1)$ and $\mathcal{IP}(0) = \mathcal{AM}(0)$ are trivial.) We comment that it is common to shorthand $\mathcal{AM}(2)$ by $\mathcal{AM}$, which is indeed inconsistent with the convention of using $\mathcal{IP}$ as shorthand of $\mathcal{IP}(\texttt{poly})$.

The fact that $\mathcal{IP}(O(f)) = \mathcal{IP}(f)$ is proved by establishing an analogous result for $\mathcal{AM}(\cdot)$ demonstrates the advantage of the public-coin setting for the study of interactive proofs. A similar phenomenon occurs when establishing that the IP-hierarchy equals an analogous two-sided error hierarchy (see Exercise 9.8).

9.1.4.4 Something completely different

We stress that although we have relaxed the requirements from the verification procedure (by allowing it to interact with the prover, toss coins, and risk some (bounded) error probability), we did not restrict the validity of its assertions by assumptions concerning the potential prover. This should be contrasted with other notions of proof systems, such as computationally-sound ones (see §9.1.5.2), in which the validity of the verifier's assertions depends on assumptions concerning the potential prover(s).

9.1.5 On computationally bounded provers: an overview

Recall that our definition of interactive proofs (i.e., Definition 9.1) makes no reference to the computational abilities of the potential prover. This fact has two conflicting consequences:

1. The completeness condition does not provide any upper bound on the complexity of the corresponding proving strategy (which convinces the verifier to accept valid assertions).

2. The soundness condition guarantees that, regardless of the computational effort spend by a cheating prover, the verifier cannot be fooled to accept invalid assertions (with probability exceeding the soundness error).

Note that providing an upper-bound on the complexity of the (prescribed) prover strategy P of a specific interactive proof system (P, V) only strengthens the claim that (P, V) is a proof system for the corresponding set (of valid assertions). We stress that the prescribed prover strategy is referred to only in the completeness condition (and is irrelevant to the soundness condition). On the other hand, relaxing the definition of interactive proofs such that soundness holds only for a specific class of cheating prover strategies (rather than for all cheating prover strategies) weakens the corresponding claim. In this advanced section we consider both possibilities.

Teaching note: Indeed, this is an advanced subsection, which is best left for independent reading. It merely provides an overview of the various notions, and the reader is directed to the chapter's notes for further detail (i.e., pointers to the relevant literature).

9.1.5.1 How powerful should the prover be?

Suppose that a set S is in $\mathcal{IP}$. This means that there exists a verifier V that can be convinced to accept any input in S but cannot be fooled to accept any input not in S (except with small probability). One may ask how powerful should a prover be such that it can convince the verifier V to accept any input in S. Note that Proposition 9.5 asserts that an optimal prover strategy (for convincing any fixed verifier V) can be implemented in polynomial-space, and that we cannot expect any better for a generic set in $\mathcal{PSPACE} = \mathcal{IP}$ (because the emulation of the interaction of V with any optimal prover strategy yields a decision procedure for the set). Still, we may seek better upper-bounds on the complexity of some prover strategy that convinces a specific verifier, which in turn corresponds to a specific set S. More interestingly, considering all possible verifiers that give rise to interactive proof systems for S, we ask what is the minimum power required from a prover that satisfies the completeness requirement with respect to one of these verifiers?

We stress that, unlike the case of computationally-sound proof systems (see §9.1.5.2), we do not restrict the power of the prover in the soundness condition, but rather consider the minimum complexity of provers meeting the completeness condition. Specifically, we are interested in *relatively efficient* provers that meet the completeness condition. The term "relatively efficient prover" has been given three different interpretations, which are briefly surveyed next.

1. A prover is considered *relatively efficient* if, when given an auxiliary input (in addition to the common input in S), it works in (probabilistic) polynomial-time. Specifically, in case $S \in \mathcal{NP}$, the auxiliary input maybe an NP-proof that the common input is in the set. Still, even in this case the interactive proof need not consist of the prover sending the auxiliary input to the verifier; for example, an alternative procedure may allow the prover to be zero-knowledge (see Construction 9.10).

   This interpretation is adequate and in fact crucial for applications in which such an auxiliary input is available to the otherwise polynomial-time parties. Typically, such auxiliary input is available in cryptographic applications in which parties wish to prove in (zero-knowledge) that they have correctly conducted some computation. In these cases, the NP-proof is just the transcript of the computation by which the claimed result has been generated, and thus the auxiliary input is available to the proving party.

2. A prover is considered *relatively efficient* if it can be implemented by a probabilistic polynomial-time oracle machine with oracle access to the set S itself. Note that the prover in Construction 9.3 has this property (and see also Exercise 9.10).

   This interpretation generalizes the notion of self-reducibility of NP-proof systems. Recall that by self-reducibility of an NP-set (or rather of the corresponding NP-proof system) we mean that the search problem of finding an NP-witness is polynomial-time reducible to deciding membership in the set (cf. Definition 2.14). Here we require that implementing the prover strategy (in the relevant interactive proof) be polynomial-time reducible to deciding membership in the set.

3. A prover is considered *relatively efficient* if it can be implemented by a probabilistic machine that runs in time that is polynomial in the deterministic complexity of the set. This interpretation relates the time-complexity of convincing a "lazy person" (i.e., a verifier) to the time-complexity of determining the truth (i.e., deciding membership in the set).

   Hence, in contrast to the first interpretation, which is adequate in settings where assertions are generated along with their NP-proofs, the current interpretation is adequate in settings in which the prover is given only the assertion and has to find a proof to it by itself (before trying to convince a lazy verifier of its validity).

9.1.5.2 Computational-soundness

Relaxing the soundness condition such that it only refers to relatively-efficient ways of trying to fool the verifier (rather than to all possible ways) yields a fundamentally different notion of a proof system. Assertions proved in such a system are not necessarily correct; they are correct only if the potential cheating prover does not exceed the presumed complexity limits. As in §9.1.5.1, the notion of "relative efficiency" can be given different interpretations, the most popular one being that the cheating prover strategy can be implemented by a (non-uniform) family of polynomial-size circuits. The latter interpretation coincides with the first interpretation used in §9.1.5.1 (i.e., a probabilistic polynomial-time strategy that is given an auxiliary input (of polynomial length)). Specifically, in this case, the soundness condition is replaced by the following computational soundness condition that asserts that it is infeasible to fool the verifier into accepting false statements. Formally:

> *For every prover strategy that is implementable by a family of polynomial-size circuits $\{C_n\}$, and every sufficiently long $x \in \{0,1\}^* \setminus S$, the probability that V accepts x when interacting with $C_{|x|}$ is less than $1/2$.*

As in case of standard soundness, the computational-soundness error can be reduced by repetitions. We warn, however, that unlike in the case of standard soundness (where both sequential and parallel repetitions will do), the computational-soundness error cannot *always* be reduced by parallel repetitions.

It is common and natural to consider proof systems in which the prover strategies considered both in the completeness and soundness conditions satisfy the same notion of relative efficiency. Protocols that satisfy these conditions with respect to the foregoing interpretation are called arguments. We mention that argument systems may be more efficient (e.g., in terms of their communication complexity) than interactive proof systems.

9.2 Zero-Knowledge Proof Systems

Standard mathematical proofs are believed to yield (extra) knowledge and not merely establish the validity of the assertion being proved; that is, it is commonly believed that (good) proofs provide a deeper understanding of the theorem being proved. At the technical level, an NP-proof of membership in some set $S \in \mathcal{NP} \setminus \mathcal{P}$ yields something (i.e., the NP-proof itself) that is hard to compute (even when assuming that the input is in S). For example, a 3-coloring of a graph constitutes an NP-proof that the graph is 3-colorable, but it yields information (i.e., the coloring) that seems infeasible to compute (when given an arbitrary 3-colorable graph).

A natural question that arises is whether or not proving an assertion always requires giving away some extra knowledge. The setting of interactive proof systems enables a negative answer to this fundamental question: In contrast to NP-proofs, which seem to yield a lot of knowledge, zero-knowledge (interactive) proofs yield no knowledge at all; that is, *zero-knowledge proofs are both convincing and yet yield nothing beyond the validity of the assertion being proved.* For example, a zero-knowledge proof of 3-colorability does not yield any information about the graph (e.g., partial information about a 3-coloring) that is infeasible to compute from the graph itself. Thus, zero-knowledge proofs exhibit an extreme contrast between being convincing (of the validity of a assertion) and teaching anything on top of the validity of the assertion.

Needless to say, the notion of zero-knowledge proofs is fascinating (e.g., since it differentiates proof-verification from learning). Still, the reader may wonder whether such a phenomenon is desirable, because in many settings we do care to learn as much as possible (rather than learn as little as possible). However, in other settings (most notably in cryptography), we may actually wish to limit the gain that other parties may obtained from a proof (and, in particular, limit this gain to the minimal level of being convinced in the validity of the assertion). Indeed, the applicability of zero-knowledge proofs in the domain of cryptography is vast; they are typically used as a tool for forcing (potentially malicious) parties to behave according to a predetermined protocol (without having them reveal their own private inputs). The interested reader is referred to discussions in §C.4.3.3 and §C.7.3.2 (and to detailed treatments in [90, 91]). We also mention that, in addition to their direct applicability in Cryptography, zero-knowledge proofs serve

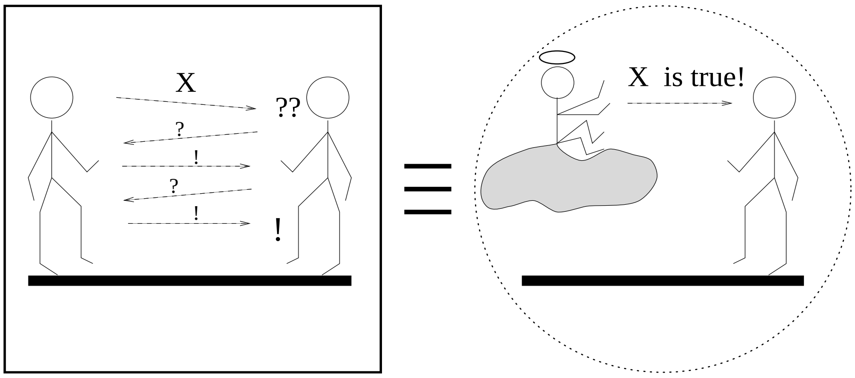


Figure 9.2: Zero-knowledge proofs – an illustration.

as a good bench-mark for the study of various questions regarding cryptographic protocols.

Teaching note: We believe that the treatment of zero-knowledge proofs provided in this section suffices for the purpose of a course in complexity theory. For an extensive treatment of zero-knowledge proofs, the interested reader is referred to [90, Chap. 4].

9.2.1 Definitional Issues

Loosely speaking, zero-knowledge proofs are proofs that yield nothing beyond the validity of the assertion; that is, a verifier obtaining such a proof only gains conviction in the validity of the assertion. This is formulated by saying that anything that can be feasibly obtained from a zero-knowledge proof is also feasibly computable from the (valid) assertion itself. The latter formulation follows the simulation paradigm, which is discussed next.

9.2.1.1 A wider perspective: the simulation paradigm

In defining zero-knowledge proofs, we view the verifier as a potential adversary that tries to gain knowledge from the (prescribed) prover.[16] We wish to state that no (feasible) adversary strategy for the verifier can gain anything from the prover (beyond conviction in the validity of the assertion). The question addressed here is how to formulate the "no gain" requirement.

Let us consider the desired formulation from a wide perspective. A key question regarding the modeling of security concerns is how to express the intuitive requirement that an adversary "gains nothing substantial" by deviating from the prescribed behavior of an honest user. The answer is that *the adversary* gains nothing *if whatever it can obtain by unrestricted adversarial behavior can be obtained*

[16]Recall that when defining a proof system (e.g., an interactive proof system), we view the prover as a potential adversary that tries to fool the (prescribed) verifier (into accepting invalid assertions).

within essentially the same computational effort by a benign (or prescribed) *behavior.* The definition of the "benign behavior" captures what we want to achieve in terms of security, and is specific to the security concern to be addressed. For example, in the context of zero-knowledge, *a benign behavior is any computation that is based* (only) *on the assertion itself* (while assuming that the latter is valid). Thus, a zero-knowledge proof is an interactive proof in which no feasible adversarial verifier strategy can obtain from the interaction more than a "benign party" (which believes the assertion) can obtain from the assertion itself.

The foregoing interpretation of "gaining nothing" means that any feasible adversarial behavior can be "simulated" by a benign behavior (and thus there is no gain in the former). This line of reasoning is called the simulation paradigm, and is pivotal to many definitions in cryptography (e.g., it underlies the definitions of security of encryption schemes and cryptographic protocols); for further details see Appendix C.

9.2.1.2 The basic definitions

We turn back to the concrete task of defining zero-knowledge. Firstly, we comment that zero-knowledge is a property of some prover strategies; actually, more generally, zero-knowledge is a property of some strategies. Fixing any strategy (e.g., a prescribed prover), we consider what can be gained (i.e., computed) by an *arbitrary feasible adversary* (e.g., a verifier) *that interacts with the aforementioned fixed strategy* on a common input taken from a predetermined set (in our case the set of valid assertions). This gain is compared against what can be computed by an *arbitrary feasible algorithm* (called a simulator) that is only given the input itself. The fixed strategy is zero-knowledge if the "computational power" of these two (fundamentally different settings) is essentially equivalent. Details follow.

The formulation of the zero-knowledge condition refers to two types of probability ensembles, where each ensemble associates a single probability distribution to each relevant input (e.g., a valid assertion). Specifically, in the case of interactive proofs, the first ensemble represents the output distribution of the verifier after interacting with the specified prover strategy P (on some common input), where the verifier is employing an arbitrary efficient strategy (not necessarily the specified one). The second ensemble represents the output distribution of some probabilistic polynomial-time algorithm (which is only given the corresponding input (and does not interact with anyone)). The basic paradigm of zero-knowledge asserts that for every ensemble of the first type there exist a "similar" ensemble of the second type. The specific variants differ by the interpretation given to the notion of *similarity.* The most strict interpretation, leading to perfect zero-knowledge, is that similarity means equality.

Definition 9.7 (perfect zero-knowledge, over-simplified):[17] *A prover strategy, P,*

[17]In the actual definition one relaxes the requirement in one of the following two ways. The first alternative is allowing A^* to run for *expected* (rather than strict) polynomial-time. The second alternative consists of allowing A^* to have no output with probability at most $1/2$ and considering the value of its output conditioned on it having output at all. The latter alternative implies the former, but the converse is not known to hold.

is said to be perfect zero-knowledge over a set S *if for every probabilistic polynomial-time verifier strategy,* V^**, there exists a probabilistic polynomial-time algorithm,* A^**, such that*

$$(P,V^*)(x) \equiv A^*(x)\,, \qquad \text{for every } x \in S$$

where $(P,V^*)(x)$ *is a random variable representing the output of verifier* V^* *after interacting with the prover* P *on common input* x*, and* $A^*(x)$ *is a random variable representing the output of algorithm* A^* *on input* x*.*

We comment that any set in co$\mathcal{RP}$ has a perfect zero-knowledge proof system in which the prover keeps silence and the verifier decides by itself. The same holds for $\mathcal{BPP}$ provided that we relax the definition of interactive proof system to allow two-sided error. Needless to say, our focus is on non-trivial proof systems; that is, proof systems for sets outside of $\mathcal{BPP}$.

A somewhat more relaxed interpretation (of the notion of similarity), leading to almost-perfect zero-knowledge (a.k.a statistical zero-knowledge), is that similarity means statistical closeness (i.e., negligible difference between the ensembles). The most liberal interpretation, leading to the standard usage of the term zero-knowledge (and sometimes referred to as computational zero-knowledge), is that similarity means computational indistinguishability (i.e., failure of any efficient procedure to tell the two ensembles apart). Combining the foregoing discussion with the relevant definition of computational indistinguishability (i.e., Definition C.5), we obtain the following definition.

Definition 9.8 (zero-knowledge, somewhat simplified): *A prover strategy,* P*, is said to be* zero-knowledge *over a set* S *if for every probabilistic polynomial-time verifier strategy,* V^**, there exists a probabilistic polynomial-time* simulator*,* A^**, such that for every probabilistic polynomial-time* distinguisher*,* D*, it holds that*

$$d(n) \stackrel{\text{def}}{=} \max_{x\in S\cap\{0,1\}^n}\{|\mathsf{Pr}[D(x,(P,V^*)(x))\!=\!1] - \mathsf{Pr}[D(x,A^*(x))\!=\!1]|\}$$

is a negligible function.[18] *We denote by* $\mathcal{ZK}$ *the class of sets having zero-knowledge interactive proof systems.*

Definition 9.8 is a simplified version of the actual definition, which is presented in Appendix C.4.2. Specifically, in order to guarantee that zero-knowledge is preserved under sequential composition it is necessary to slightly augment the definition (by providing V^* and A^* with the same value of an arbitrary (poly$(|x|)$-bit long) auxiliary input). Other definitional issues and related notions are briefly discussed in Appendix C.4.4.

[18] That is, d vanishes faster that the reciprocal of any positive polynomial (i.e., for every positive polynomial p and for sufficiently large n, it holds that $d(n) < 1/p(n)$). Needless to say, $d(n) \stackrel{\text{def}}{=} 0$ if $S\cap\{0,1\}^n = \emptyset$.

On the role of randomness and interaction. It can be shown that only sets in $\mathcal{BPP}$ have zero-knowledge proofs in which the verifier is deterministic (see Exercise 9.13). The same holds for deterministic provers, provided that we consider "auxiliary-input" zero-knowledge (as in Definition C.9). It can also be shown that only sets in $\mathcal{BPP}$ have zero-knowledge proofs in which a single message is sent (see Exercise 9.14). Thus, both randomness and interaction are essential to the non-triviality of zero-knowledge proof systems. (For further details, see [90, Sec. 4.5.1].)

Advanced Comment: Knowledge Complexity. Zero-knowledge is the lowest level of a knowledge-complexity hierarchy which quantifies the "knowledge revealed in an interaction." Specifically, the knowledge complexity of an interactive proof system may be defined as the minimum number of oracle-queries required in order to efficiently simulate an interaction with the prover. (See [89, Sec. 2.3.1] for references.)

9.2.2 The Power of Zero-Knowledge

When faced with a definition as complex (and seemingly self-contradictory) as the definition of zero-knowledge, one should indeed wonder whether the definition can be met (in a non-trivial manner).[19] It turns out that the existence of non-trivial zero-knowledge proofs is related to the existence of intractable problems in $\mathcal{NP}$. In particular, we will show that if one-way functions exist then every NP-set has a zero-knowledge proof system. (For the converse, see [90, Sec. 4.5.2] or [227].) But first, we demonstrate the non-triviality of zero-knowledge by a presenting a simple (perfect) zero-knowledge proof system for a specific NP-set that is not known to be in $\mathcal{BPP}$. In this case we make no intractability assumptions (yet, the result is significant only if $\mathcal{NP}$ is not contained in $\mathcal{BPP}$).

9.2.2.1 A simple example

> *A story not found in the Odyssey refers to the not so famous Labyrinth of the Island of Aeaea. The Sorceress Circe, daughter of Helius, challenged godlike Odysseus to traverse the Labyrinth from its North Gate to its South Gate. Canny Odysseus doubted whether such a path existed at all and asked beautiful Circe for a proof, to which she replied that if she showed him a path this would trivialize for him the challenge of traversing the Labyrinth. "Not necessarily," clever Odysseus replied, "you can use your magic to transport me to a random place in the labyrinth, and then guide me by a random walk to a gate of my choice. If we repeat this enough times then I'll be convinced that there is a labyrinth-path between the two gates, while you will not reveal to me such a path." "Indeed," wise Circe thought to herself, "showing this mortal a random path from a random location in the labyrinth to*

[19] Recall that any set in $\mathcal{BPP}$ has a trivial zero-knowledge (two-sided error) proof system in which the verifier just determines membership by itself. Thus, the issue is the existence of zero-knowledge proofs for sets outside $\mathcal{BPP}$.

> *the gate he chooses will not teach him more than his taking a random walk from that gate."*

The foregoing story illustrates the main idea underlying the zero-knowledge proof for Graph Isomorphism presented next. Recall that the set of pairs of isomorphic graphs is not known to be in $\mathcal{BPP}$, and thus the straightforward NP-proof system (in which the prover just supplies the isomorphism) may not be zero-knowledge. Furthermore, assuming that Graph Isomorphism is not in $\mathcal{BPP}$, this set has no zero-knowledge NP-proof system. Still, as we shall shortly see, this set does have a zero-knowledge interactive proof system.

Construction 9.9 (zero-knowledge proof for Graph Isomorphism):

- Common Input: *A pair of graphs,* $G_1=(V_1,E_1)$ *and* $G_2=(V_2,E_2)$.

  If the input graphs are indeed isomorphic, then we let ϕ *denote an arbitrary isomorphism between them; that is,* ϕ *is a 1-1 and onto mapping of the vertex set* V_1 *to the vertex set* V_2 *such that* $\{u,v\} \in E_1$ *if and only if* $\{\phi(v),\phi(u)\} \in E_2$.

- Prover's first Step (P1): *The prover selects a random isomorphic copy of* G_2*, and sends it to the verifier. Namely, the prover selects at random, with uniform probability distribution, a permutation* π *from the set of permutations over the vertex set* V_2*, and constructs a graph with vertex set* V_2 *and edge set*

  $$E \stackrel{\rm def}{=} \{\{\pi(u),\pi(v)\} : \{u,v\}\in E_2\} .$$

  The prover sends (V_2,E) *to the verifier.*

- Motivating Remark: *If the input graphs are isomorphic, as the prover claims, then the graph sent in Step P1 is isomorphic to both input graphs. However, if the input graphs are* not *isomorphic then no graph can be isomorphic to both of them.*

- Verifier's first Step (V1): *Upon receiving a graph,* $G' = (V',E')$*, from the prover, the verifier asks the prover to show an isomorphism between* G' *and one of the input graphs, chosen at random by the verifier. Namely, the verifier uniformly selects* $\sigma \in \{1,2\}$*, and sends it to the prover (who is supposed to answer with an isomorphism between* G_σ *and* G'*).*

- Prover's second Step (P2): *If the message,* σ*, received from the verifier equals* 2 *then the prover sends* π *to the verifier. Otherwise (i.e.,* $\sigma \neq 2$*), the prover sends* $\pi \circ \phi$ *(i.e., the composition of* π *on* ϕ*, defined as* $\pi \circ \phi(v) \stackrel{\rm def}{=} \pi(\phi(v))$*) to the verifier.*

  (Indeed, the prover treats any $\sigma \neq 2$ *as* $\sigma = 1$*. Thus, in the analysis we shall assume, without loss of generality, that* $\sigma \in \{1,2\}$ *always holds.)*

- Verifier's second Step (V2): *If the message, denoted* ψ*, received from the prover is an isomorphism between* G_σ *and* G' *then the verifier outputs 1, otherwise it outputs 0.*

The verifier strategy in Construction 9.9 is easily implemented in probabilistic polynomial-time. If the prover is given an isomorphism between the input graphs as auxiliary input, then also the prover's program can be implemented in probabilistic polynomial-time. The motivating remark justifies the claim that Construction 9.9 constitutes an interactive proof system for the set of pairs of isomorphic graphs. Thus, we focus on establishing the zero-knowledge property.

We consider first the special case in which the verifier actually follows the prescribed strategy (and selects σ at random, and in particular obliviously of the graph G' it receives). The view of this verifier can be easily simulated by selecting σ and ψ at random, constructing G' as a random isomorphic copy of G_σ (via the isomorphism ψ), and outputting the triple (G', σ, ψ). Indeed (even in this case), the simulator behaves differently from the prescribed prover (which selects G' as a random isomorphic copy of G_2, via the isomorphism π), but its output distribution is identical to the verifier's view in the real interaction. However, the foregoing description assumes that the verifier follows the prescribed strategy, while in general the verifier may (adversarially) select σ depending on the graph G'. Thus, a slightly more complicated simulation (described next) is required.

A general clarification may be in place. Recall that we wish to simulate the interaction of an arbitrary verifier strategy with the prescribed prover. Thus, this simulator must depend on the corresponding verifier strategy, and indeed we shall describe the simulator while referring to such a generic verifier strategy. Formally, this means that the simulator's program incorporates the program of the corresponding verifier strategy. Actually, the following simulator uses the generic verifier strategy as a subroutine.

Turning back to the specific protocol of Construction 9.9, the basic idea is that simulator tries to guess σ and completes a simulation if its guess turns out to be correct. Specifically, the simulator selects $\tau \in \{1, 2\}$ uniformly (hoping that the verifier will later select $\sigma = \tau$), and constructs G' by randomly permuting G_τ (and thus being able to present an isomorphism between G_τ and G'). Recall that the simulator is analyzed only on yes-instances (i.e., the input graphs G_1 and G_2 are isomorphic). The point is that if G_1 and G_2 are isomorphic, then the graph G' does not yield any information regarding the simulator's guess (i.e., τ).[20] Thus, the value σ selected by the adversarial verifier may depend on G' but not on τ, which implies that $\mathsf{Pr}[\sigma = \tau] = 1/2$. In other words, the simulator's guess (i.e., τ) is correct (i.e., equals σ) with probability $1/2$. Now, if the guess is correct then the simulator can produce an output that has the correct distribution, and otherwise the entire process is repeated.

Digest: a few useful conventions. We highlight three conventions that were either used (implicitly) in the foregoing analysis or can be used to simplify the description of (this and/or) other zero-knowledge simulators.

1. Without loss of generality, we may assume that the cheating verifier strategy is implemented by a *deterministic* polynomial-size circuit (or, equivalently,

[20] Indeed, this observation is identical to the observation made in the analysis of the soundness of Construction 9.3.

by a deterministic polynomial-time algorithm with an auxiliary input).[21]

This is justified by fixing any outcome of the verifier's coins, and observing that our (uniform) simulation of the various (residual) deterministic strategies yields a simulation of the original probabilistic strategy.

2. Without loss of generality, it suffices to consider cheating verifiers that (only) output their view of the interaction (i.e., the common input, their internal coin tosses, and the messages that they have received). In other words, it suffices to simulate the view that cheating verifiers have of the real interaction.

   This is justified by noting that the final output of any verifier can be obtained from its view of the interaction, where the complexity of the transformation is upper-bounded by the complexity of the verifier's strategy.

3. Without loss of generality, it suffices to construct a "weak simulator" that produces output with some noticeable[22] probability such that whenever an output is produced it is distributed "correctly" (i.e., similarly to the distribution occuring in real interactions with the prescribed prover).

   This is justified by repeatedly invoking such a weak simulator (polynomially) many times and using the first output produced by any of these invocations. Note that by using an adequate number of invocations, we fail to produce an output with negligible probability. Furthermore, note that a simulator that fails to produce output with negligible probability can be converted to a simulator that always produces an output, while incurring a negligible statistic deviation in the output distribution.

9.2.2.2 The full power of zero-knowledge proofs

The zero-knowledge proof system presented in Construction 9.9 refers to one specific NP-set that is not known to be in $\mathcal{BPP}$. It turns out that, under reasonable assumptions, zero-knowledge can be used to prove membership in *any* NP-set. Intuitively, it suffices to establish this fact for a single NP-complete set, and thus we focus on presenting a zero-knowledge proof system for the set of 3-colorable graphs.

It is easy to prove that a given graph G is 3-colorable by just presenting a 3-coloring of G (and the same holds for membership in any set in $\mathcal{NP}$), but this NP-proof is not a zero-knowledge proof (unless $\mathcal{NP} \subseteq \mathcal{BPP}$). In fact, assuming $\mathcal{NP} \not\subseteq \mathcal{BPP}$, graph 3-colorability has no zero-knowledge NP-proof system. Still, as we shall shortly see, graph 3-colorability does have a zero-knowledge interactive proof system. This proof system will be described while referring to "boxes" in which information can be hidden and later revealed. Such boxes can be implemented using one-way functions (see, e.g., Theorem 9.11).

[21] This observation is not crucial, but it does simplify the analysis (by eliminating the need to specify a sequence of coin tosses in each invocation of the verifier's strategy).

[22] Recall that a probability is called noticeable if it is greater than the reciprocal of some positive polynomial (in the relevant parameter).

Construction 9.10 (Zero-knowledge proof of 3-colorability, abstract description): *The description refers to abstract non-transparent boxes that can be perfectly locked and unlocked such that these boxes perfectly hide their contents while being locked.*

- Common Input: *A simple graph* $G = (V, E)$.
- Prover's first step: *Let* ψ *be a 3-coloring of* G. *The prover selects a random permutation,* π, *over* $\{1, 2, 3\}$, *and sets* $\phi(v) \stackrel{\text{def}}{=} \pi(\psi(v))$, *for each* $v \in V$. *Hence, the prover forms a random relabeling of the 3-coloring* ψ. *The prover sends to the verifier a sequence of* $|V|$ *locked and non-transparent boxes such that the* v^{th} *box contains the value* $\phi(v)$.
- Verifier's first step: *The verifier uniformly selects an edge* $\{u, v\} \in E$, *and sends it to the prover.*
- Motivating Remark: *The boxes are supposed to contain a 3-coloring of the graph, and the verifier asks to inspect the colors of vertices* u *and* v. *Indeed, for the zero-knowledge condition, it is crucial that the prover only responds to pairs that correspond to edges of the graph.*
- Prover's second step: *Upon receiving an edge* $\{u, v\} \in E$, *the prover sends to the verifier the keys to boxes* u *and* v.

  For simplicity of the analysis, if the verifier sends $\{u, v\} \notin E$ *then the prover behaves as if it has received a fixed* (or random) *edge in* E, *rather than suspending the interaction, which would have been the natural thing to do.*
- Verifier's second step: *The verifier unlocks and opens boxes* u *and* v, *and accepts if and only if they contain two different elements in* $\{1, 2, 3\}$.

The verifier strategy in Construction 9.10 is easily implemented in probabilistic polynomial-time. The same holds with respect to the prover's strategy, provided that it is given a 3-coloring of G as auxiliary input. Clearly, if the input graph is 3-colorable then the verifier accepts with probability 1 when interacting with the prescribed prover. On the other hand, if the input graph is not 3-colorable, then any contents put in the boxes must be invalid with respect to at least one edge, and consequently the verifier will reject with probability at least $\frac{1}{|E|}$. Hence, the foregoing protocol exhibits a non-negligible gap in the accepting probabilities between the case of 3-colorable graphs and the case of non-3-colorable graphs. To increase the gap, the protocol may be repeated sufficiently many times (of course, using independent coin tosses in each repetition).

So far we showed that Construction 9.10 constitutes (a weak form of) an interactive proof system for Graph 3-Colorability. The point, however, is that the prescribed prover strategy is zero-knowledge. This is easy to see in the abstract setting of Construction 9.10, because all that the verifier sees in the real interaction is a sequence of boxes and a random pair of *different* colors (which is easy to simulate). Indeed, the simulation of the real interaction proceeds by presenting a sequence of boxes and providing a random pair of different colors as the contents of the two boxes indicated by the verifier. Note that the foregoing argument relies on the fact that the boxes (indicated by the verifier) correspond to vertices that are connected by an edge in the graph.

This simple demonstration of the zero-knowledge property is not possible in the digital implementation (discussed next), because in that case the boxes are not totally unaffected by their contents (but are rather affected, yet in an indistinguishable manner). Thus, the verifier's selection of the inspected edge may depend on the "outside appearance" of the various boxes, which in turn may depend (in an indistinguishable manner) on the contents of these boxes. Consequently, we cannot determine the boxes' contents after a pair of boxes are selected, and so the simple foregoing simulation is inapplicable. Instead, we simulate the interaction as follows.

1. We first guess (at random) which pair of boxes (corresponding to an edge) the verifier would ask to open, and place a random pair of distinct colors in these boxes (and garbage in the rest).[23] Then, we hand all boxes to the verifier, which asks us to open a pair of boxes (corresponding to an edge).

2. If the verifier asks for the pair that we chose (i.e., our guess is successful), then we can complete the simulation by opening these boxes. Otherwise, we try again (i.e., repeat Step 1 with a new random guess and random colors). The key observation is that if the boxes hide the contents in the sense that a box's contents is indistinguishable based on it outside appearance, then our guess will succeed with probability approximately $1/|E|$. Furthermore, in this case, the simulated execution will be indistinguishable from the real interaction.

Thus, it suffices to use boxes that hide their contents almost perfectly (rather than being perfectly opaque). Such boxes can be implemented digitally.

> **Teaching note:** Indeed, we recommend presenting and analyzing in class only the foregoing abstract protocol. It suffices to briefly comment about the digital implementation, rather than presenting a formal proof of Theorem 9.11 (which can be found in [99] (or [90, Sec. 4.4])).

Digital implementation (overview). We implement the abstract boxes (referred to in Construction 9.10) by using adequately defined commitment schemes. Loosely speaking, such a scheme is a two-phase game between a sender and a receiver such that after the first phase the sender is "committed" to a value and yet, at this stage, it is infeasible for the receiver to find out the committed value (i.e., the commitment is "hiding"). The committed value will be revealed to the receiver in the second phase and it is guaranteed that the sender cannot reveal a value other than the one committed (i.e., the commitment is "binding"). Such commitment

[23] An alternative (and more efficient) simulation consists of putting random independent colors in the various boxes, hoping that the verifier asks for an edge that is properly colored. The latter event occurs with probability (approximately) 2/3, provided that the boxes hide their contents (almost) perfectly.

schemes can be implemented assuming the existence of one-way functions (as in Definition 7.3); see §C.4.3.1.

Zero-knowledge proofs for other NP-sets. Using the fact that 3-colorability is NP-complete, one can derive (from Construction 9.10) zero-knowledge proof systems for any NP-set.[24] Furthermore, NP-witnesses can be efficiently transformed into polynomial-size circuits that implement the corresponding (prescribed zero-knowledge) prover strategies.

Theorem 9.11 (The ZK Theorem): *Assuming the existence of* (non-uniformly hard) *one-way functions, it holds that* $\mathcal{NP} \subseteq \mathcal{ZK}$. *Furthermore, every* $S \in \mathcal{NP}$ *has a* (computational) *zero-knowledge interactive proof system in which the prescribed prover strategy can be implemented in probabilistic polynomial-time, provided that it is given as auxiliary-input an NP-witness for membership of the common input in* S.

The hypothesis of Theorem 9.11 (i.e., the existence of one-way functions) seems unavoidable, because the existence of zero-knowledge proofs for "hard on the average" problems implies the existence of one-way functions (and, likewise, the existence of zero-knowledge proofs for sets outside $\mathcal{BPP}$ implies the existence of "auxiliary-input one-way functions").

Theorem 9.11 has a dramatic effect on the design of cryptographic protocols (see Appendix C). In a different vein we mention that, under the same assumption, any interactive proof can be transformed into a zero-knowledge one. (This transformation, however, does not necessarily preserve the complexity of the prover.)

Theorem 9.12 (The ultimate ZK Theorem): *Assuming the existence of* (non-uniformly hard) *one-way functions, it holds that* $\mathcal{IP} = \mathcal{ZK}$.

Loosely speaking, Theorem 9.12 can be proved by recalling that $\mathcal{IP} = \mathcal{AM}(\texttt{poly})$ and modifying any public-coin protocol as follows: the modified prover sends commitments to its messages rather than the messages themselves, and once the original interaction is completed it proves (in zero-knowledge) that the corresponding transcript would have been accepted by the original verifier. Indeed, the latter assertion is of the "NP type", and thus the zero-knowledge proof system guaranteed in Theorem 9.11 can be invoked for proving it.

Reflection. The proof of Theorem 9.11 uses the fact that 3-colorability is NP-complete in order to obtain a zero-knowledge proofs for any set in $\mathcal{NP}$ by using such a protocol for 3-colorability (i.e., Construction 9.10). Thus, an NP-completeness result is used here in a "positive" way; that is, in order to construct something rather than in order to derive a ("negative") hardness result (cf., Section 2.2.4).[25]

---

[24] Actually, we should either rely on the fact that the standard Karp-reductions are invertible in polynomial time or on the fact that the 3-colorability protocol is actually zero-knowledge with respect to auxiliary inputs (as in Definition C.9).

[25] Historically, the proof of Theorem 9.11 was probably the first positive application of NP-completeness. Subsequent positive uses of completeness results have appeared in the context of

Perfect and Statistical Zero-Knowledge. The foregoing results, which refer to computational zero-knowledge proof systems, should be contrasted with the known results regarding the complexity of statistical zero-knowledge proof systems: Statistical zero-knowledge proof systems exist only for sets in $\mathcal{IP}(2) \cap \mathrm{co}\mathcal{IP}(2)$, and thus are unlikely to exist for all NP-sets. On the other hand, the class Statistical Zero-Knowledge is known to contain some seemingly hard problems, and turns out to have interesting complexity theoretic properties (e.g., being closed under complementation, and having very natural complete problems). The interested reader is referred to [226].

9.2.3 Proofs of Knowledge – a parenthetical subsection

Teaching note: Technically speaking, this topic belongs to Section 9.1, but its more interesting demonstrations refer to zero-knowledge proofs of knowledge – hence its current positioning.

Loosely speaking, "proofs of knowledge" are interactive proofs in which the prover asserts "knowledge" of some object (e.g., a 3-coloring of a graph), and not merely its existence (e.g., the existence of a 3-coloring of the graph, which in turn is equivalent to the assertion that the graph is 3-colorable). Note that the entity asserting knowledge is actually the prover's strategy, which is an automated computing device, hereafter referred to as a machine. This raises the question of what do we mean by saying that a *machine knows something*.

9.2.3.1 Abstract reflections

Any standard dictionary suggests several meanings for the verb `to know`, but these are typically phrased with reference to the notion of *awareness*, a notion which is certainly inapplicable in the context of machines. Instead, we should look for a *behavioristic* interpretation of the verb `to know`. Indeed, it is reasonable to link knowledge with the ability to do something (e.g., the ability to write down whatever one knows). Hence, we may say that a machine knows a string α if it *can* output the string α. But this seems as total non-sense too: a machine has a well defined output – either the output equals α or it does not, so what can be meant by saying that *a machine can do something*?

Interestingly, a sound interpretation of the latter phrase does exist. Loosely speaking, by saying that *a machine can do something* we mean that the machine can be *easily modified* such that it (or rather its modified version) does whatever is claimed. More precisely, this means that there exists an *efficient* machine that, using the original machine as a black-box (or given its code as an input), outputs whatever is claimed.

Technically speaking, using a machine as a black-box seems more appealing when the said machine is interactive (i.e., implements an interactive strategy). Indeed, this will be our focus here. Furthermore, conceptually speaking, whatever

interactive proofs (see the proof of Theorem 9.4), probabilistically checkable proofs (see the proof of Theorem 9.16), and the study of statistical zero-knowledge (cf. [226]).

a machine knows (or does not know) is its own business, whereas what can be of interest and reference *to the outside* is whatever can be deduced about the knowledge of a machine by interacting with it. Hence, we are interested in proofs of knowledge (rather than in mere knowledge).

9.2.3.2 A concrete treatment

For sake of simplicity let us consider a concrete question: *how can a machine prove that it knows a 3-coloring of a graph?* An obvious way is just sending the 3-coloring to the verifier. Yet, we claim that applying the protocol in Construction 9.10 (i.e., the zero-knowledge proof system for 3-Colorability) is an alternative way of proving knowledge of a 3-coloring of the graph.

The definition of a *verifier of knowledge of 3-coloring* refers to any possible prover strategy and links the ability to "extract" a 3-coloring (of a given graph) from such a prover to the probability that this prover convinces the verifier. That is, the definition postulates the existence of an efficient universal way of "extracting" a 3-coloring of a given graph by using any prover strategy that convinces this verifier to accept this graph with probability 1 (or, more generally, with some noticeable probability). On the other hand, we should no expect this extractor to obtain much from prover strategies that fail to convince the verifier (or, more generally, convince it with negligible probability). A robust definition should allow a smooth transition between these two extremes (and in particular between provers that convince the verifier with noticeable probability and those that convince it with negligible probability). Such a definition should also support the intuition by which the following strategy of Alice is zero-knowledge: *Alice sends Bob a 3-coloring of a given graph provided that Bob has successfully convinced her that he knows this coloring.*[26] We stress that the zero-knowledge property of Alice's strategy should hold regardless of the proof-of-knowledge system used for proving Bob's knowledge of a 3-coloring.

Loosely speaking, we say that a strategy, V, constitutes a verifier for knowledge of 3-coloring if, for any prover strategy P, the complexity of extracting a 3-coloring of G when using P as a "black box"[27] is inversely proportional to the probability that V is convinced by P (to accept the graph G). Namely, the extraction of the 3-coloring is done by an oracle machine, called an extractor, that is given access to the strategy P (i.e., the function specifying the message that P sends in response to any sequence of messages it may receive). We require that the (*expected*) *running time of the extractor, on input G and oracle access to P, be inversely related* (by a factor polynomial in $|G|$) *to the probability that P convinces V to accept G*. In particular, if P always convinces V to accept G, then the extractor runs in expected polynomial-time. The same holds in case P convinces V to accept with noticeable probability. On the other hand, if P never convinces V to accept, then nothing is required of the extractor. We stress that the latter special cases do not suffice for

[26]For simplicity, the reader may consider graphs that have a unique 3-coloring (up-to a relabeling). In general, we refer here to instances that have unique solution (cf. Section 6.2.3), which arise naturally in some (cryptographic) applications.

[27]Indeed, one may consider also non-black-box extractors.

a satisfactory definition; see discussion in [90, Sec. 4.7.1].

Proofs of knowledge, and in particular zero-knowledge proofs of knowledge, have many applications to the design of cryptographic schemes and cryptographic protocols (see, e.g., [90, 91]). These are enabled by the following general result.

Theorem 9.13 (Theorem 9.11, revisited): *Assuming the existence of* (non-uniformly hard) *one-way functions, any NP-relation has a zero-knowledge proof of knowledge* (of a corresponding NP-witnesses)*. Furthermore, the prescribed prover strategy can be implemented in probabilistic polynomial-time, provided it is given such an NP-witness.*

9.3 Probabilistically Checkable Proof Systems

Teaching note: Probabilistically checkable proof (PCP) systems may be viewed as a restricted type of interactive proof systems in which the prover is memoryless and responds to each verifier message as if it were the first such message. This perspective creates a tighter link with previous sections, but is somewhat contrived. Indeed, such a memoryless prover may be viewed as a static object that the verifier may query at locations of its choice. But then it is more appealing to present the model using the (more traditional) terminology of oracle machines rather than using (and degenerating) the terminology of interactive machines (or strategies).

Probabilistically checkable proof systems can be viewed as standard (deterministic) proof systems that are augmented with a probabilistic procedure capable of evaluating the validity of the assertion by examining few locations in the alleged proof. Actually, we focus on the latter probabilistic procedure, which in turn implies the existence of a deterministic verification procedure (obtained by going over all possible random choices of the probabilistic procedure and making the adequate examinations).

Modeling such probabilistic verification procedures, which may examine few locations in the alleged proof, requires providing these procedures with direct access to the individual bits of the alleged proof (so that they need not scan the proof bit-by-bit). Thus, the alleged proof is a string, as in the case of a traditional proof system, but the (probabilistic) verification procedure is given direct access to individual bits of this string.

We are interested in *probabilistic verification procedures that access only few locations in the proof, and yet are able to make a meaningful probabilistic verdict regarding the validity of the alleged proof.* Specifically, the verification procedure should accept any valid proof (with probability 1), but rejects with probability at least 1/2 any alleged proof for a false assertion. Such probabilistic verification procedures are called probabilistically checkable proof (PCP) systems.

The fact that one can (meaningfully) evaluate the correctness of proofs by examining few locations in them is indeed amazing and somewhat counter-intuitive. Needless to say, such proofs must be written in a somewhat non-standard format, because standard proofs cannot be verified without reading them in full (since a flaw may be due to a single improper inference). In contrast, proofs for a PCP system tend to be very redundant; they consist of superfluously many pieces of information (about the claimed assertion), but their correctness can be (meaningfully) evaluated by *checking the consistency of a randomly chosen collection of few related pieces.* We stress that by a "meaningful evaluation" we mean rejecting alleged proofs of false assertions with constant probability (rather than with probability that is inversely proportional to the length of the alleged proof).

The main complexity measure associated with PCPs is indeed their query complexity. Another complexity measure of natural concern is the length of the proofs being employed, which in turn is related to the randomness complexity of the system. The randomness complexity of PCPs plays a key role in numerous applications (e.g., in composing PCP systems as well as when applying PCP systems to derive inapproximability results), and thus we specify this parameter rather than the proof length.

Teaching note: Indeed, PCP systems are most famous for their role in deriving numerous inapproximability results (see Section 9.3.3), but our view is that the latter is merely one extremely important application of the fundamental notion of a PCP system. Our presentation is organized accordingly.

9.3.1 Definition

Loosely speaking, a probabilistically checkable proof system consists of a probabilistic polynomial-time verifier having access to an oracle that represents an alleged proof (in redundant form). Typically, the verifier accesses only few of the oracle bits, and these bit positions are determined by the outcome of the verifier's coin tosses. As in the case of interactive proof systems, it is required that if the assertion holds then the verifier always accepts (i.e., when given access to an adequate oracle); whereas, if the assertion is false then the verifier must reject with probability at least $\frac{1}{2}$, no matter which oracle is used. The basic definition of the PCP setting is given in Part (1) of the following definition. Yet, the complexity measures introduced in Part (2) are of key importance for the subsequent discussions.

Definition 9.14 (Probabilistically Checkable Proofs – PCP):

1. *A* probabilistically checkable proof system (PCP) for a set S *is a probabilistic polynomial-time oracle machine, called* verifier *and denoted* V*, that satisfies the following two conditions:*
   - Completeness: *For every* $x \in S$ *there exists an oracle* π_x *such that, on input* x *and access to oracle* π_x*, machine* V *always accepts* x*.*
   - Soundness: *For every* $x \notin S$ *and every oracle* π*, on input* x *and access to oracle* π*, machine* V *rejects* x *with probability at least* $\frac{1}{2}$*.*
2. *We say that a probabilistically checkable proof system has* query complexity $q:\mathbb{N}\to\mathbb{N}$ *if, on any input of length* n*, the verifier makes at most* $q(n)$ *oracle*

queries.[28] *Similarly, the* randomness complexity $r:\mathbb{N}\to\mathbb{N}$ *upper-bounds the number of coin tosses performed by the verifier on a generic n-bit long input.*

For integer functions r and q, we denote by $\mathcal{PCP}(r,q)$ the class of sets having probabilistically checkable proof systems of randomness complexity r and query complexity q. For sets of integer functions, R and Q,

$$\mathcal{PCP}(R,Q) \stackrel{\rm def}{=} \bigcup_{r\in R\,,\,q\in Q} \mathcal{PCP}(r,q)\,.$$

The error probability (in the soundness condition) of PCP systems can be reduced by successive applications of the proof system. In particular, repeating the process for k times, reduces the probability that the verifier is fooled by a false assertion to 2^{-k}, whereas all complexities increase by at most a factor of k. Thus, PCP systems of non-trivial query-complexity (cf. Section 9.3.2) provide a trade-off between the number of locations examined in the proof and the confidence in the validity of the assertion.

We note that the oracle π_x referred to in the completeness condition of a PCP system constitutes a proof in the standard mathematical sense. Indeed any PCP system yields a standard proof system (with respect to a verification procedure that scans all possible outcomes of V's internal coin tosses and emulates all the corresponding checks). Furthermore, the oracles in PCP systems of logarithmic randomness-complexity constitute NP-proofs (see Exercise 9.15). However, the oracles of a PCP system have the *extra remarkable property* of enabling a lazy verifier to toss coins, take its chances and "assess" the validity of the proof without reading all of it (but rather by reading a tiny portion of it). Potentially, this allows the verifier to examine very few bits of an NP-proof and even utilize very long proofs (i.e., of super-polynomial length).

Adaptive versus non-adaptive verifiers. Definition 9.14 allows the verifier to be adaptive; that is, the verifier may determine its queries based on the answers it has received to previous queries (in addition to their dependence on the input and on the verifier's internal coin tosses). In contrast, non-adaptive verifiers determine all their queries based solely on their input and internal coin tosses. Note that q adaptive (binary) queries can be emulated by $\sum_{i=1}^{q} 2^{i-1} < 2^q$ non-adaptive (binary) queries. We comment that most constructions of PCP systems use non-adaptive verifiers, and in fact in many sources PCP systems are defined as non-adaptive.

Randomness versus proof length. Fixing a verifier V, we say that location i (in the oracle) is relevant to input x if there exists a computation of V on input x in which location i is queried (i.e., there exists ω and π such that, on input x, randomness ω and access to the oracle π, the verifier queries location i). The effective proof length of V is the smallest function $\ell:\mathbb{N}\to\mathbb{N}$ such that for every input x there are at most $\ell(|x|)$ locations (in the oracle) that are relevant to x.

[28]As usual in complexity theory, the oracle answers are binary values (i.e., either 0 or 1).

We claim that the effective proof length of any PCP system is closely related to its randomness (and query) complexity. On one hand, *if the PCP system has randomness-complexity r and query-complexity q, then its effective proof length is upper-bounded by* 2^{r+q}, whereas a bound of $2^r \cdot q$ holds for non-adaptive systems (see Exercise 9.15). Thus, *PCP systems of logarithmic randomness complexity have effective proof length that is polynomial*, and hence yield NP-proof systems. On the other hand, in some sense, the randomness complexity of a PCP system can be upper-bounded by the logarithm of the (effective) length of the proofs employed (provided we allow non-uniform verifiers; see Exercise 9.16).

On the role of randomness. The PCP Theorem (i.e., $\mathcal{NP} \subseteq \mathcal{PCP}(\texttt{log}, O(1))$) asserts that a meaningful probabilistic evaluation of proofs is possible based on a constant number of examined bits. We note that, unless $\mathcal{P} = \mathcal{NP}$, such a phenomena is impossible when requiring the verifier to be deterministic. Firstly, note that $\mathcal{PCP}(0, O(1)) = \mathcal{P}$ holds (as a special case of $\mathcal{PCP}(r,q) \subseteq \text{DTIME}(2^{2^r q+r} \cdot \text{poly})$; see Exercise 9.17). Secondly, as shown in Exercise 9.19, $\mathcal{P} \neq \mathcal{NP}$ implies that $\mathcal{NP}$ is not contained in $\mathcal{PCP}(o(\log), o(\log))$. Lastly, assuming that not all NP-sets have NP-proof systems that employs proofs of length ℓ (e.g., $\ell(n) = n$), it follows that if $2^{r(n)}q(n) < \ell(n)$ then $\mathcal{PCP}(r,q)$ does not contain $\mathcal{NP}$ (see Exercise 9.17 again).

9.3.2 The Power of Probabilistically Checkable Proofs

The celebrated PCP Theorem asserts that $\mathcal{NP} = \mathcal{PCP}(\texttt{log}, O(1))$, and this result is indeed the focus of the current section. But before getting to it we make several simple observations regarding the PCP Hierarchy.

We first note that $\mathcal{PCP}(\texttt{poly}, 0)$ equals co$\mathcal{RP}$, whereas $\mathcal{PCP}(0, \texttt{poly})$ equals $\mathcal{NP}$. It is easy to prove an upper bound on the non-deterministic time complexity of sets in the PCP hierarchy (see Exercise 9.17):

Proposition 9.15 (upper-bounds on the power of PCPs): *For every polynomially bounded integer function r, it holds that $\mathcal{PCP}(r, \texttt{poly}) \subseteq \text{NTIME}(2^r \cdot \texttt{poly})$. In particular, $\mathcal{PCP}(\texttt{log}, \texttt{poly}) \subseteq \mathcal{NP}$.*

The focus on PCP systems of logarithmic randomness complexity reflects an interest in PCP systems that utilize proof oracles of polynomial length (see discussion in Section 9.3.1). We stress that such PCP systems (i.e., $\mathcal{PCP}(\texttt{log}, q)$) are NP-proof systems with a (potentially amazing) extra property: the validity of the assertion can be "probabilistically evaluated" by examining a (small) portion (i.e., $q(n)$ bits) of the proof. Thus, for any fixed polynomially bounded function q, a result of the form

$$\mathcal{NP} \subseteq \mathcal{PCP}(\texttt{log}, q) \tag{9.6}$$

is interesting (because it applies also to NP-sets having witnesses of length exceeding q). Needless to say, the smaller q – the better. The PCP Theorem asserts the amazing fact by which q can be made a constant.

Theorem 9.16 (The PCP Theorem): $\mathcal{NP} \subseteq \mathcal{PCP}(\texttt{log}, O(1))$.

Thus, probabilistically checkable proofs in which the verifier tosses only logarithmically many coins and makes only a constant number of queries exist for every set in $\mathcal{NP}$. This constant is essentially three (see §9.3.4.1). Before reviewing the proof of Theorem 9.16, we make a couple of comments.

Efficient transformation of NP-witnesses to PCP oracles: The proof of Theorem 9.16 is constructive in the sense that it allows to efficiently transform any NP-witness (for an instance of a set in $\mathcal{NP}$) into an oracle that makes the PCP verifier accept (with probability 1). That is, *for every* (NP-witness relation) *$R \in \mathcal{PC}$ there exists a PCP verifier V as in Theorem 9.16 and a polynomial-time computable function π such that for every $(x, y) \in R$ the verifier V always accepts the input x when given oracle access to the proof $\pi(x, y)$* (i.e., $\mathsf{Pr}[V^{\pi(x,y)}(x) = 1] = 1$). Recalling that the latter oracles are themselves NP-proofs, it follows that NP-proofs can be transformed into NP-proofs that offer a trade-off between the portion of the proof being read and the confidence it offers. Specifically, for every $\varepsilon > 0$, if one is willing to tolerate an error probability of ε then it suffices to examine $O(\log(1/\varepsilon))$ bits of the (transformed) NP-proof. Indeed (as discussed in Section 9.3.1), these bit locations need to be selected at random.

The foregoing strengthening of Theorem 9.16 offers a wider range of applications than Theorem 9.16 itself. Indeed, Theorem 9.16 itself suffices for "negative" applications such as establishing the infeasibility of certain approximation problems (see Section 9.3.3). But for "positive" applications (see §9.3.4.2), typically some user (or a real entity) will be required to actually construct the PCP-oracle, and in such cases the strengthening of Theorem 9.16 will be useful.

A characterization of NP: Combining Theorem 9.16 with Proposition 9.15 we obtain the following characterization of $\mathcal{NP}$.

Corollary 9.17 (The PCP characterization of NP): $\mathcal{NP} = \mathcal{PCP}(\texttt{log}, O(1))$.

Road-map for the proof of the PCP Theorem: Theorem 9.16 is a culmination of a sequence of remarkable works, each establishing meaningful and increasingly stronger versions of Eq. (9.6). A presentation of the full proof of Theorem 9.16 is beyond the scope of the current work (and is, in our opinion, unsuitable for a basic course in complexity theory). Instead, we present an overview of the original proof (see §9.3.2.2) as well as of an alternative proof (see §9.3.2.3), which was found more than a decade later. We will start, however, by presenting a weaker result that is used in both proofs of Theorem 9.16 and is also of independent interest. This weaker result (see §9.3.2.1) asserts that *every NP-set has a PCP system with constant query-complexity* (albeit with polynomial randomness complexity); that is, $\mathcal{NP} \subseteq \mathcal{PCP}(\texttt{poly}, O(1))$.

Teaching note: In our opinion, presenting in class any part of the proof of the PCP Theorem should be given low priority. In particular, presenting the connections between PCP and the complexity of approximation should be given a higher priority. As for relative priorities among the following three subsections, we strongly recommend giving §9.3.2.1 the highest priority, because it offers a direct demonstration of the power of PCPs. As for the two alternative proofs of the PCP Theorem itself, our recommendation depends on the intended goal. On one hand, for the purpose of merely giving a taste of the ideas involved in the proof, we prefer an overview of the original proof (provided in §9.3.2.2). On the other hand, for the purpose of actually providing a full proof, we definitely prefer the new proof (which is only outlined in §9.3.2.3).

9.3.2.1 Proving that $\mathcal{NP} \subseteq \mathcal{PCP}(\texttt{poly}, O(1))$

The fact that every NP-set has a PCP system with constant query-complexity (regardless of its randomness-complexity) already testifies to the power of PCP systems. It asserts that *probabilistic verification of proofs is possible by inspecting very few locations in a* (potentially huge) *proof*. Indeed, the PCP systems presented next utilize exponentially long proofs, but they do so while inspecting these proofs at a constant number of (randomly selected) locations.

We start with a brief overview of the construction. We first note that it suffices to construct a PCP for proving the satisfiability of a given system of quadratic equations over GF(2), because this problem is NP-complete (see Exercise 2.25).[29] For an input consisting of a system of quadratic equations with n variables, the oracle (of this PCP) is supposed to provide the evaluation of all quadratic expressions (in these n variables) at some fixed assignment to these variables. This assignment is supposed to satisfy the system of quadratic equations that is given as input. We distinguish two tables in the oracle: the first table corresponding to all 2^n linear expressions and the second table to all 2^{n^2} quadratic expressions. Each table is tested for self-consistency (via a "linearity test"), and the two tables are tested to be consistent with each other (via a "matrix-equality" test, which utilizes "self-correction"). Finally, we test that the assignment encoded in these tables satisfies the quadratic system that is given as input. This is done by taking a random linear combination of the quadratic equations that appear in the quadratic system, and obtaining the value assigned to the corresponding quadratic expression by the aforementioned tables (again, via self-correction). The key point is that each of the foregoing tests utilizes a constant number of Boolean queries, and has time (and randomness) complexity that is polynomial in the size of the input. Details follow.

Teaching note: The following text refers to notions such as the Hadamard encoding, testing and self-correction, which appear in other parts of this work (see, e.g., §E.1.1.2, Section 10.1.2. and §7.2.1.1, respectively). While a wider perspective (provided in the aforementioned parts) is always useful, the current text is self-contained.

[29] Here and elsewhere, we denote by GF(2) the 2-element field.

The starting point. We construct a PCP system for the set of satisfiable quadratic equations over GF(2). The input is a sequence of such equations over the variables $x_1, ..., x_n$, and the proof oracle consist of two parts (or tables), which are supposed to provide information regarding some satisfying assignment $\tau = \tau_1 \cdots \tau_n$ (also viewed as an n-ary vector over GF(2)). The first part, denoted T_1, is supposed to provide a Hadamard encoding of the said satisfying assignment; that is, for every $\alpha \in \mathrm{GF}(2)^n$ this table is supposed to provide the inner product mod 2 of the n-ary vectors α and τ (i.e., $T_1(\alpha)$ is supposed to equal $\sum_{i=1}^n \alpha_i \tau_i$). The second part, denoted T_2, is supposed to provide all linear combinations of the values of the $\tau_i \tau_j$'s; that is, for every $\beta \in \mathrm{GF}(2)^{n^2}$ (viewed as an n-by-n matrix over GF(2)), the value of $T_2(\beta)$ is supposed to equal $\sum_{i,j} \beta_{i,j} \tau_i \tau_j$. (Indeed T_1 is contained in T_2, because $\sigma^2 = \sigma$ for any $\sigma \in \mathrm{GF}(2)$.) The PCP verifier will use the two tables for checking that the input (i.e., a sequence of quadratic equations) is satisfied by the assignment that is encoded in the two tables. Needless to say, these tables may not be a valid encoding of any n-ary vector (let alone one that satisfies the input), and so the verifier also needs to check that the encoding is (close to being) valid. We will focus on this task first.

Testing the Hadamard Code. Note that T_1 is supposed to encode a linear function; that is, there must exist some $\tau = \tau_1 \cdots \tau_n \in \mathrm{GF}(2)^n$ such that $T_1(\alpha) = \sum_{i=1}^n \tau_i \alpha_i$ holds for every $\alpha = \alpha_1 \cdots \alpha_n \in \mathrm{GF}(2)^n$. This can be tested by selecting uniformly $\alpha', \alpha'' \in \mathrm{GF}(2)^n$ and checking whether $T_1(\alpha') + T_1(\alpha'') = T_1(\alpha' + \alpha'')$, where $\alpha' + \alpha''$ denotes addition of vectors over GF(2). The analysis of this natural tester turns out to be quite complex. Nevertheless, it is indeed the case that any table that is 0.02-far from being linear is rejected with probability at least 0.01 (see Exercise 9.20), where T is ε-far from being linear if T disagrees with any linear function f on more than an ε fraction of the domain (i.e., $\mathsf{Pr}_r[T(r) \neq f(r)] > \varepsilon$).

By repeating the linearity test for a constant number of times, we may reject each table that is 0.02-far from being a codeword of the Hadamard Code with probability at least 0.99. Thus, using a constant number of queries, the verifier rejects any T_1 that is 0.02-far from being a Hadamard encoding of any $\tau \in \mathrm{GF}(2)^n$, and likewise rejects any T_2 that is 0.02-far from being a Hadamard encoding of any $\tau' \in \mathrm{GF}(2)^{n^2}$. We may thus assume that T_1 (resp., T_2) is 0.02-close to the Hadamard encoding of some τ (resp., τ').[30] (Needless to say, this does *not* mean that τ' equals the outer produce of τ with itself.)

In the rest of the analysis, we fix $\tau \in \mathrm{GF}(2)^n$ and $\tau' \in \mathrm{GF}(2)^{n^2}$, and denote the Hadamard encoding of τ (resp., τ') by $f_\tau : \mathrm{GF}(2)^n \to \mathrm{GF}(2)$ (resp., $f_{\tau'} : \mathrm{GF}(2)^{n^2} \to \mathrm{GF}(2)$). Recall that T_1 (resp., T_2) is 0.02-close to f_τ (resp., $f_{\tau'}$).

Self-correction of the Hadamard Code. Suppose that T is ε-close to a linear function $f : \mathrm{GF}(2)^m \to \mathrm{GF}(2)$ (i.e., $\mathsf{Pr}_r[T(r) \neq f(r)] \leq \varepsilon$). Then, we can recover the value of f at any desired point x, by making two (random) queries to T.

[30] Note that τ (resp., τ') is uniquely determined by T_1 (resp., T_2), because every two different linear functions $\mathrm{GF}(2)^m \to \mathrm{GF}(2)$ agree on exactly half of the domain (i.e., the Hadamard code has relative distance 1/2).

Specifically, for a uniformly selected $r \in \mathrm{GF}(2)^m$, we use the value $T(x+r)-T(r)$. Note that the probability that we recover the correct value is at least $1-2\varepsilon$, because $\mathsf{Pr}_r[T(x+r)-T(r)=f(x+r)-f(r)] \geq 1-2\varepsilon$ and $f(x+r)-f(r)=f(x)$ by linearity of f. (Needless to say, for $\varepsilon < 1/4$, the function T cannot be ε-close to two different linear functions.)[31] Thus, assuming that T_1 is 0.02-close to f_τ (resp., T_2 is 0.02-close to $f_{\tau'}$) we may correctly recover (i.e., with error probability 0.04) the value of f_τ (resp., $f_{\tau'}$) at any desired point by making 2 queries to T_1 (resp., T_2). This process is called self-correction (cf., e.g., §7.2.1.1).

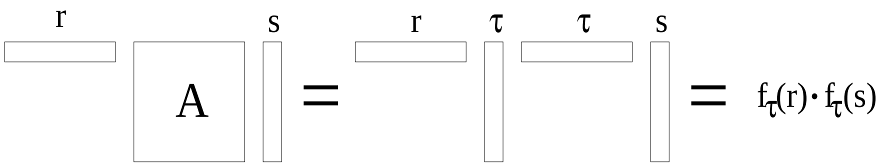


Figure 9.3: Detail for testing consistency of linear and quadratic forms.

Checking consistency of f_τ and $f_{\tau'}$. Suppose that we are given access to $f_\tau : \mathrm{GF}(2)^n \to \mathrm{GF}(2)$ and $f_{\tau'} : \mathrm{GF}(2)^{n^2} \to \mathrm{GF}(2)$, where $f_\tau(\alpha) = \sum_i \tau_i \alpha_i$ and $f_{\tau'}(\alpha') = \sum_{i,j} \tau'_{i,j}\alpha'_{i,j}$, and that we wish to verify that $\tau'_{i,j} = \tau_i\tau_j$ for every $i,j \in \{1,...,n\}$. In other words, we are given a (somewhat weird) encoding of two matrices, $A = (\tau_i\tau_j)_{i,j}$ and $A' = (\tau'_{i,j})_{i,j}$, and we wish to check whether or not these matrices are identical. It can be shown (see Exercise 9.22) that if $A \neq A'$ then $\mathsf{Pr}_{r,s}[r^\top As \neq r^\top A's] \geq 1/4$, where r and s are uniformly distributed n-ary vectors. Note that, in our case (where $A = (\tau_i\tau_j)_{i,j}$ and $A' = (\tau'_{i,j})_{i,j}$), it holds that $r^\top As = \sum_j(\sum_i r_i\tau_i\tau_j)s_j = f_\tau(r)f_\tau(s)$ (see Figure 9.3) and $r^\top A's = \sum_j(\sum_i r_i\tau'_{i,j})s_j = f_{\tau'}(rs^\top)$, where $rs^\top$ is the outer-product of s and r. Thus, (for $(\tau_i\tau_j)_{i,j} \neq (\tau'_{i,j})_{i,j}$) we have $\mathsf{Pr}_{r,s}[f_\tau(r)f_\tau(s) \neq f_{\tau'}(rs^\top)] \geq 1/4$.

Recall, however, that we do not have direct access to the functions f_τ and $f_{\tau'}$, but rather to tables (i.e., T_1 and T_2) that are 0.02-close to these functions. Still, using self-correction, we can obtain the values of f_τ and $f_{\tau'}$ at any desired point, with very high probability. Actually, when implementing the foregoing consistency test it suffices to use self-correction for $f_{\tau'}$, because we use the values of f_τ at two independently and uniformly distributed points in $\mathrm{GF}(2)^n$ (i.e., r,s) but the value $f_{\tau'}$ is required at $rs^\top$, which is not uniformly distributed in $\mathrm{GF}(2)^{n^2}$. Thus, we test the consistency of f_τ and $f_{\tau'}$ by selecting uniformly $r,s \in \mathrm{GF}(2)^n$ and $R \in \mathrm{GF}(2)^{n^2}$, and checking that $T_1(r)T_1(s) = T_2(rs^\top + R) - T_2(R)$.

By repeating the aforementioned (self-corrected) consistency test for a constant number of times, we may reject an inconsistent pair of tables with probability at least 0.99. Thus, in the rest of the analysis, we may assume that $(\tau_i\tau_j)_{i,j} = (\tau'_{i,j})_{i,j}$.

[31]Indeed, this fact follows from the self-correction argument, but a simpler proof merely refers to the fact that the Hadamard code has relative distance 1/2.

Checking that τ satisfies the quadratic system. Suppose that we are given access to f_τ and $f_{\tau'}$ as in the foregoing (where, in particular, $\tau' = \tau\tau^\top$). A key observation is that if τ does not satisfy a system of (quadratic) equations then, with probability 1/2, it does not satisfy a random linear combination of these equations. Thus, in order to check whether τ satisfies the quadratic system (which is given as input), we create a single quadratic equation by taking such a random linear combination, and check whether this quadratic equation is satisfied by τ. The punch-line is that *testing whether τ satisfies the quadratic equation $Q(x) = \sigma$ amounts to testing whether $f_{\tau'}(Q) = \sigma$*. Again, the actual checking is implemented by using self-correction (of the table T_2).

This completes the description of the verifier. Note that this verifier performs a constant number of codeword tests for the Hadamard Code, and a constant number of consistency and satisfiability tests, where each of the latter involves self-correction of the Hadamard Code. Each of the individual tests utilizes a constant number of queries (ranging between two and four) and uses randomness that is quadratic in the number of variables (and linear in the number of equations in the input). Thus, the query-complexity is a constant and the randomness-complexity is at most quadratic in the length of the input (quadratic system). Clearly, if the input quadratic system is satisfiable (by some τ), then the verifier accepts the corresponding tables T_1 and T_2 (i.e., $T_1 = f_\tau$ and $T_2 = f_{\tau\tau^\top}$) with probability 1. On the other hand, if the input quadratic system is unsatisfiable, then any pair of tables (T_1, T_2) will be rejected with constant probability (by one of the foregoing tests). It follows that $\mathcal{NP} \subseteq \mathcal{PCP}(q, O(1))$, where q is a quadratic polynomial.

Reflection. Indeed, the actual test of the satisfiability of the quadratic system that is given as input is facilitated by the fact that a satisfying assignment is encoded (in the oracle) in a very redundant manner, which fits the final test of satisfiability. But then the burden of testing moves to checking that this encoding is indeed valid. In fact, most of the tests performed by the foregoing verifier are aimed at verifying the validity of the encoding. Such a test of validity (of encoding) may be viewed as a test of consistency between the various parts of the encoding. All these themes are present also in more advanced constructions of PCP systems.

9.3.2.2 Overview of the first proof of the PCP Theorem

The original proof of the PCP Theorem (Theorem 9.16) consists of three main conceptual steps, *which we briefly sketch first and further discuss later.*

1. Constructing a (non-adaptive) PCP system for $\mathcal{NP}$ having *logarithmic randomness and polylogarithmic query complexity*; that is, this PCP has the desired randomness complexity and a very low (but non-constant) query complexity. Furthermore, this proof system has additional properties that enable proof composition as in the following Step 3.

2. Constructing a PCP system for $\mathcal{NP}$ having *polynomial randomness and constant query complexity*; that is, this PCP has the desired (constant) query

complexity but its randomness complexity is prohibitingly high. (Indeed, we showed such a construction in §9.3.2.1.) Furthermore, this proof system too has additional properties enabling proof composition as in Step 3.

3. The proof composition paradigm:[32] In general, this paradigm allows to compose two proof systems such that the "inner" one is used for probabilistically verifying the acceptance criteria of the "outer" verifier. The aim is to conduct this ("composed") verification using much fewer queries than the query complexity of the "outer" proof system. In particular, the inner verifier cannot afford to read its input, which makes composition more subtle than the term suggests.

   Loosely speaking, the *outer* verifier should be robust in the sense that its soundness condition guarantee that with high probability the oracle answers are "far" from satisfying the residual decision predicate (rather than merely not satisfy it). (Furthermore, the latter predicate, which is well-defined by the non-adaptive nature of the outer verifier, must have a circuit of size bounded by a polynomial in the number of queries.) The *inner* verifier is given oracle access to its input and is charged for each query made to it, but is only required to reject with high probability inputs that are far from being valid (and, as usual, accept inputs that are valid). That is, the inner verifier is actually a verifier of proximity.

   Composing two such PCPs yields a new PCP for $\mathcal{NP}$, where the new proof oracle consists of the proof oracle of the "outer" system and a sequence of proof oracles for the "inner" system (one "inner" proof per each possible random-tape of the "outer" verifier). The resulting verifier selects coins for the outer-verifier and uses the corresponding "inner" proof in order to verify that the outer-verifier would have accepted under this choice of coins. Note that such a choice of coins determines locations in the "outer" proof that the outer-verifier would have inspected, and the combined verifier provides the inner-verifier with oracle access to these locations (which the inner-verifier considers as its input) as well as with oracle access to the corresponding "inner" proof (which the inner-verifier considers as its proof-oracle).

   Note that composing an outer-verifier of randomness-complexity r' and query-complexity q' with an inner-verifier of randomness-complexity r'' and query-complexity q'' yields a PCP of randomness-complexity $r(n) = r'(n)+r''(q'(n))$ and query-complexity $q(n) = q''(q'(n))$, because $q'(n)$ represents the length of the input (oracle) that is accessed by the inner-verifier. Recall that the outer-verifier is non-adaptive, and thus if the inner-verifier is non-adaptive (resp., robust) then so is the verifier resulting from the composition, which is important in case we wish to compose the latter verifier with another inner-verifier.

In particular, the proof system of Step 1 is composed with itself [using $r'(n) = r''(n) = O(\log n)$ and $q'(n) = q''(n) = \mathrm{poly}(\log n)$] yielding a PCP system (for

[32] Our presentation of the composition paradigm follows [34], rather than the original presentation of [15, 14].

$\mathcal{NP}$) of randomness-complexity $r(n) = r'(n) + r''(q'(n)) = O(\log n)$ and query-complexity $q(n) = q''(q'(n)) = \text{poly}(\log\log n)$. Composing the latter system (used as an "outer" system) with the PCP system of Step 2, yields a PCP system (for $\mathcal{NP}$) of randomness-complexity $r(n)+\text{poly}(q(n)) = O(\log n)$ and query-complexity $O(1)$, thus establishing the PCP Theorem.

A more detailed overview – the plan. The foregoing description uses two (non-trivial) PCP systems and refers to additional properties such as robustness and verification of proximity. A PCP system of polynomial randomness-complexity and constant query-complexity (as postulated in Step 2) is outlined in §9.3.2.1. We thus start by discussing the notions of verifying proximity and being robust, while demonstrating their applicability to the said PCP. Finally, we outline the other PCP system that is used (i.e., the one postulated in Step 1).

PCPs of Proximity. Recall that a standard PCP verifier gets an explicit input and is given oracle access to an alleged proof (for membership of the input in a predetermined set). In contrast, a PCP of proximity verifier is given (direct) *access to two oracles, one representing an input and the other being an alleged proof, and its queries to both oracles are counted in its query-complexity.* Typically, the query-complexity of this verifier is lower than the length of the input oracle, and hence this verifier cannot afford reading the entire input and cannot be expected to make absolute statements about it. Indeed, instead of deciding whether or not the input is in a predetermined set, the verifier is only required to distinguish the case that the input is in the set from the case that the input is *far* from the set (where far means being at *relative* Hamming distance at least 0.01 (or any other small constant)).

For example, consider a variant of the system of §9.3.2.1 in which the quadratic system is fixed[33] and the verifier needs to determine whether the assignment appearing in the input oracle satisfies the said system or is far from any assignment that satisfies it. We use a proof oracle is as in §9.3.2.1, and a PCP verifier of proximity that proceeds as in §9.3.2.1 and in addition perform a proximity test to verify that the input oracle is close to the assignment encoded in the proof oracle. Specifically, the verifier reads a uniformly selected bit of the input oracle and compares this value to the self-corrected value obtained from the proof oracle (i.e., for a uniformly selected $i \in \{1, ..., n\}$, we compare the i^{th} bit of the input oracle to the self-correction of the value $T_1(0^{i-1}10^{n-i})$, obtained from the proof oracle).

Robust PCPs. Composing an "outer" PCP verifier with an "inner" PCP verifier of proximity makes sense provided that the *outer* verifier rejects in a "robust" manner. That is, the soundness condition of a robust verifier requires that (with probability at least 1/2) the oracle answers are *far* from any sequence that is acceptable by the residual predicate (rather than merely that the answers are rejected by this predicate). Indeed, if the outer verifier is (non-adaptive and) robust, then

[33]Indeed, in our applications the quadratic system will be "known" to the ("inner") verifier, because it is determined by the ("outer") verifier.

it suffices that the inner verifier distinguish (with the help of an adequate proof) answers that are valid from answers that are far from being valid.

For example, if robustness is defined as referring to *relative constant distance* (which is indeed the case), then the PCP of §9.3.2.1 (as well as any PCP of constant query complexity) is trivially robust. However, we will not care about the robustness of this PCP, because we only use this PCP as an inner verifier in proof composition. In contrast, we will care about the robustness of PCPs that are used as outer verifiers (e.g., the PCP presented next).

Teaching note: Unfortunately, the construction of a PCP of logarithmic randomness and polylogarithmic query complexity for NP involves many technical details. Furthermore, obtaining a robust version of this PCP is beyond the scope of the current text. Thus, the following description should be viewed as merely providing a flavor of the underlying ideas.

PCP of logarithmic randomness and polylogarithmic query complexity for NP. We focus on showing that $NP \subseteq \mathcal{PCP}(f,f)$, for $f(n) = \text{poly}(\log n)$, and the claimed result will follow by a relatively minor modification (discussed afterwards). The proof system underlying $NP \subseteq \mathcal{PCP}(f,f)$ is based on an arithmetization of 3CNF formulae, which is different from the one used in §9.1.3.2 (for constructing an interactive proof system for co$\mathcal{NP}$). We start by describing this arithmetization, and later outline the PCP system that is based on it.

In the current arithmetization, the names of the variables (resp., clauses) of a 3CNF formula ϕ are represented by binary strings of logarithmic (in $|\phi|$) length, and a *generic* variable (resp., clause) of ϕ is represented by a logarithmic number of *new variables*, which are assigned values in a finite field $\mathrm{F} \supset \{0,1\}$. Indeed, throughout the rest of the description, we refer to the arithmetic operations of this finite field F (which will have cardinality $\text{poly}(|\phi|)$). The (structure of the) 3CNF formula $\phi(x_1,...,x_n)$ is represented by a Boolean function $C_\phi : \{0,1\}^{O(\log n)} \to \{0,1\}$ such that $C_\phi(\alpha,\beta_1,\beta_2,\beta_3) = 1$ if and only if, for $i = 1,2,3$, the i^{th} literal in the α^{th} clause of ϕ has index $\beta_i = (\gamma_i,\sigma_i)$, which is viewed as a variable name augmented by its sign. Thus, for every $\alpha \in \{0,1\}^{\log|\phi|}$ there is a unique $(\beta_1,\beta_2,\beta_3) \in \{0,1\}^{3\log 2n}$ such that $C_\phi(\alpha,\beta_1,\beta_2,\beta_3) = 1$ holds. Next, we consider a multi-linear extension of C_ϕ over F, denoted Φ; that is, Φ is the (unique) multi-linear polynomial that agrees with C_ϕ on $\{0,1\}^{O(\log n)} \subset \mathrm{F}^{O(\log n)}$.

Turning to the PCP, we first note that the verifier can reduce the original 3SAT-instance ϕ to the aforementioned arithmetic instance Φ; that is, on input a 3CNF formula ϕ, the verifier first constructs C_ϕ and Φ (as in Exercise 7.12). Part of the proof oracle for this verifier is viewed as function $A : \mathrm{F}^{\log n} \to \mathrm{F}$, which is supposed to be a multi-linear extension of a truth assignment that satisfies ϕ (i.e., for every $\gamma \in \{0,1\}^{\log n} \equiv [n]$, the value $A(\gamma)$ is supposed to be the value of the γ^{th} variable in such an assignment). Thus, we wish to check whether, for every $\alpha \in \{0,1\}^{\log|\phi|}$, it holds that

$$\sum_{\beta_1\beta_2\beta_3 \in \{0,1\}^{3\log 2n}} \Phi(\alpha,\beta_1,\beta_2,\beta_3) \cdot \prod_{i=1}^{3} (1 - A'(\beta_i)) = 0 \qquad (9.7)$$

where $A'(\beta)$ is the value of the β^{th} literal under the (variable) assignment A; that is, for $\beta = (\gamma, \sigma)$, where $\gamma \in \{0,1\}^{\log n}$ is a variable name and $\sigma \in \{0,1\}$ indicates the literal's type (i.e., whether the variable is negated), it holds that $A'(\beta) = (1-\sigma) \cdot A(\gamma) + \sigma \cdot (1 - A(\gamma))$. Thus, Eq. (9.7) holds if and only if the α^{th} clause is satisfied by the assignment induced by A (because $A'(\beta) = 1$ must hold for at least one of the three literals β that appear in this clause).[34]

As in §9.3.2.1, we cannot afford to verify all $|\phi|$ instances of Eq. (9.7). Furthermore, unlike in §9.3.2.1, we cannot afford to take a random linear combination of these $|\phi|$ instances either (because this requires too much randomness). Fortunately, taking a "pseudorandom" linear combination of these equations is good enough. Specifically, using an adequate (efficiently constructible) small-bias probability space (cf. §8.5.2.3) will do. Denoting such a space (of size $\text{poly}(|\phi| \cdot |F|)$ and bias at most 1/6) by $S \subset \mathrm{F}^{|\phi|}$, we may select uniformly $(s_1, ..., s_{|\phi|}) \in S$ and check whether

$$\sum_{\alpha\beta_1\beta_2\beta_3 \in \{0,1\}^{\ell}} s_\alpha \cdot \Phi(\alpha, \beta_1, \beta_2, \beta_3) \cdot \prod_{i=1}^{3} (1 - A'(\beta_i)) = 0 \tag{9.8}$$

where $\ell \stackrel{\text{def}}{=} \log|\phi| + 3\log 2n$. The small-bias property guarantees that if A fails to satisfy any of the equations of type Eq. (9.7) then, with probability at least 1/3 (taken over the choice of $(s_1, ..., s_{|\phi|}) \in S$), it is the case that A fails to satisfy Eq. (9.8). Since $|S| = \text{poly}(|\phi| \cdot |F|)$ rather that $|S| = 2^{|\phi|}$, we can select a sample in S using $O(\log|\phi|)$ coin tosses. Thus, we have reduced the original problem to checking whether, for a random $(s_1, ..., s_{|\phi|}) \in S$, Eq. (9.8) holds.

Assuming (for a moment) that A is a low-degree polynomial, we can probabilistically verify Eq. (9.8) by applying a "summation test" (as in the interactive proof for co$\mathcal{NP}$); that is, we refer to stripping the ℓ binary summations in iterations, where in each iteration the verifier obtains a corresponding univariate polynomial and instantiates it at a random point. Indeed, the verifier obtains the relevant univariate polynomials by making adequate queries (which specify the entire sequence of choices made so far in the summation test).[35] Note that after stripping the ℓ summations, the verifier end-ups with an expression that contains three unknown values of A', which it may obtain by making corresponding queries to A. The summation test involves tossing $\ell \cdot \log|\mathrm{F}|$ coins and making $(\ell+3) \cdot O(\log|\mathrm{F}|)$ Boolean queries (which correspond to ℓ queries that are each answered by a univariate polynomial of constant degree (over F), and three queries to A (each answered by an element of F)). Soundness of the summation test follows by setting $|F| \gg O(\ell)$, where $\ell = O(\log|\phi|)$.

Recall, however, that we may not assume that A is a multi-variate polynomial of low degree. Instead, we must check that A is indeed a multi-variate polynomial of

[34]Note that, for this α there exists a unique triple $(\beta_1, \beta_2, \beta_3) \in \{0,1\}^{3\log 2n}$ such that $\Phi(\alpha, \beta_1, \beta_2, \beta_3) \neq 0$. This triple $(\beta_1, \beta_2, \beta_3)$ encodes the literals appearing in the α^{th} clause, and this clause is satisfied by A if and only if $\exists i \in [3]$ s.t. $A'(\beta_i) = 1$.

[35]The query will also contain a sequence $(s_1, ..., s_{|\phi|}) \in S$, selected at random (by the verifier) and fixed for the rest of the process.

low degree (or rather that it is close to such a polynomial), and use self-correction for retrieving the values of A (which are needed for the foregoing summation test). Fortunately, a low-degree test of complexities similar to those of the summation test does exist (and self-correction is also possible within these complexities). Thus, using a finite field F of poly$(\log(n))$ elements, the foregoing yields $\mathcal{NP} \subseteq \mathcal{PCP}(f,f)$ for $f(n) \stackrel{\text{def}}{=} O(\log(n) \cdot \log\log(n))$.

To obtain the desired PCP system of logarithmic randomness complexity, we represent the names of the original variables and clauses by $\frac{O(\log n)}{\log\log n}$-long sequences over $\{1,...,\log n\}$, rather than by logarithmically-long binary sequences. This requires using low degree polynomial extensions (i.e., polynomial of degree $(\log n)-1$), rather than multi-linear extensions. We can still use a finite field of poly$(\log(n))$ elements, and so we need only $\frac{O(\log n)}{\log\log n} \cdot O(\log\log n)$ random bits for the summation and low-degree tests. However, the number of queries (needed for obtaining the answers in these tests) grows, because now the polynomials that are involved have individual degree $(\log n) - 1$ rather than constant individual degree. This merely means that the query-complexity increases by a factor of $\frac{\log n}{\log\log n}$ (since the individual degree increases by a factor of $\log n$ but the number of variables decreases by a factor of $\log\log n$). Thus, we obtain $\mathcal{NP} \subseteq \mathcal{PCP}(\texttt{log}, q)$ for $q(n) \stackrel{\text{def}}{=} O(\log^2 n)$.

Warning: Robustness and PCP of proximity. Recall that, in order to use the latter PCP system in composition, we need to guarantee that it (or a version of it) is robust as well as to present a version that is a PCP of proximity. The latter version is relatively easy to obtain (using ideas as applied to the PCP of §9.3.2.1), whereas obtaining robustness is too complex to be described here. We comment that one way of obtaining a robust PCP system is by a generic application of a (randomness-efficient) "parallelization" of PCP systems (cf. [14]), which in turn depends heavily on highly efficient low-degree tests. An alternative approach (cf. [34]) capitalizes of the specific structure of the summation test (as well as on the evident robustness of a simple low-degree test).

Reflection. The PCP Theorem asserts a PCP system that obtains simultaneously the minimal possible randomness and query complexity (up to a multiplicative factor, assuming that $\mathcal{P} \neq \mathcal{NP}$). The foregoing construction obtains this remarkable result by combining two different PCPs: the first PCP obtains logarithmic randomness but uses poly-logarithmically many queries, whereas the second PCP uses a constant number of queries but has polynomial randomness complexity. We stress that *each of these two PCP systems is highly non-trivial and very interesting by itself.* We also highlight the fact that these PCPs are combined using a very simple composition method (which refers to auxiliary properties such as robustness and proximity testing).[36]

[36] **Advanced comment:** We comment that the composition of PCP systems that lack these extra properties is possible, but is far more cumbersome and complex. In some sense, this alternative composition involves transforming the given PCP systems to ones having properties related to robustness and proximity testing.

9.3.2.3 Overview of the second proof of the PCP Theorem

The original proof of the PCP Theorem focuses on the construction of two PCP systems that are highly non-trivial and interesting by themselves, and combines them in a natural manner. Loosely speaking, this combination (via proof composition) *preserves* the good features of each of the two systems; that is, it yields a PCP system that inherits the (logarithmic) randomness complexity of one system and the (constant) query complexity of the other. In contrast, the following alternative proof is focused at the "amplification" of PCP systems, via a gradual process of logarithmically many steps. We start with a trivial "PCP" system that has the desired complexities but rejects false assertions with probability inversely proportional to their length, and in each step we double the rejection probability while essentially maintaining the initial complexities. That is, in each step, the constant query complexity of the verifier is preserved and its randomness complexity is increased only by a constant term. Thus, the process gradually transforms an extremely weak PCP system into a remarkably strong PCP system (i.e., a PCP as postulated in the PCP Theorem).

In order to describe the aforementioned process we need to redefine PCP systems so to allow arbitrary soundness error. In fact, for technical reasons, it is more convenient to describe the process as an iterated reduction of a "constraint satisfaction" problem to itself. Specifically, we refer to systems of 2-variable constraints, which are readily represented by (labeled) graphs such that the vertices correspond to (non-Boolean) variables and the edges are associated with constraints.

Definition 9.18 (CSP with 2-variable constraints): *For a fixed finite set Σ, an instance of* CSP *consists of a graph $G = (V, E)$* (which may have parallel edges and self-loops) *and a sequence of 2-variable constraints $\Phi = (\phi_e)_{e \in E}$ associated with the edges, where each constraint has the form $\phi_e : \Sigma^2 \to \{0, 1\}$. The value of an assignment $\alpha : V \to \Sigma$ is the number of constraints satisfied by α; that is, the value of α is $|\{(u, v) \in E : \phi_{(u,v)}(\alpha(u), \alpha(v)) = 1\}|$. We denote by $\mathtt{vlt}(G, \Phi)$* (standing for violation) *the fraction of unsatisfied constraints under the best possible assignment; that is,*

$$\mathtt{vlt}(G, \Phi) = \min_{\alpha : V \to \Sigma} \left\{ \frac{|\{(u, v) \in E : \phi_{(u,v)}(\alpha(u), \alpha(v)) = 0\}|}{|E|} \right\}. \tag{9.9}$$

For various functions $\tau : \mathbb{N} \to (0, 1]$, we will consider the promise problem $\mathtt{gapCSP}_\tau^\Sigma$, having instances as in the foregoing, such that the yes*-instances are fully satisfiable instances* (i.e., $\mathtt{vlt} = 0$) *and the* no*-instances are pairs (G, Φ) for which $\mathtt{vlt}(G, \Phi) \geq \tau(|G|)$ holds, where $|G|$ denotes the number of edges in G.*

Note that $\mathtt{3SAT}$ is reducible to $\mathtt{gapCSP}_\tau^{\{1,...,7\}}$ for $\tau(m) = 1/m$; see Exercise 9.23. Our goal is to reduce $\mathtt{3SAT}$ (or rather $\mathtt{gapCSP}_\tau^{\{1,...,7\}}$) to $\mathtt{gapCSP}_c^\Sigma$, for some fixed finite Σ and constant $c > 0$. The PCP Theorem will follow by showing a simple PCP system for $\mathtt{gapCSP}_c^\Sigma$; see Exercise 9.25. (The relationship between constraint satisfaction problems and the PCP Theorem is further discussed in Section 9.3.3.) The desired reduction of $\mathtt{gapCSP}_{1/m}^\Sigma$ to $\mathtt{gapCSP}_{\Omega(1)}^\Sigma$ is obtained by iteratively applying the following reduction logarithmically many times.

Lemma 9.19 (amplifying reduction of gapCSP to itself): *For some finite* Σ *and constant* $c > 0$*, there exists a polynomial-time computable function* f *such that, for every instance* (G, Φ) *of* $\texttt{gapCSP}^\Sigma$*, it holds that* $(G', \Phi') = f(G, \Phi)$ *is an instance of* $\texttt{gapCSP}^\Sigma$ *and the two instances are related as follows:*

1. *If* $\texttt{vlt}(G, \Phi) = 0$ *then* $\texttt{vlt}(G', \Phi') = 0$*.*
2. $\texttt{vlt}(G', \Phi') \geq \min(2 \cdot \texttt{vlt}(G, \Phi), c)$*.*
3. $|G'| = O(|G|)$*.*

That is, satisfiable instances are mapped to satisfiable instances, whereas instances that violate a ν fraction of the constraints are mapped to that violate at least a $\min(2\nu, c)$ fraction of the constraints. Furthermore, the mapping increases the number of edges (in the instance) by at most a constant factor. We stress that both Φ and Φ' consists of Boolean constraints defined over Σ^2.

Proof Outline:[37] Before turning to the proof, let us highlight the difficulty that it needs to address. Specifically, the lemma asserts a "violation amplifying effect" (i.e., Items 1 and 2), while maintaining the alphabet Σ and allowing only a moderate increase in the size of the graph (i.e., Item 3). Waiving the latter requirements allows a relatively simple proof that mimics (an augmented version of)[38] the parallel repetition of the corresponding PCP. Thus, the challenge is significantly decreasing the "size blow-up" that arises from parallel repetition and maintaining a fixed alphabet. The first goal (i.e., Item 3) calls for a suitable derandomization, and indeed we shall use the Expander Random Walk Generator (of Section 8.5.3). Those who read §9.3.2.2 may guess that the second goal (i.e., fixed alphabet) can be handled using the proof composition paradigm. (The rest of the overview is intended to be understood also by those who did not read Section 8.5.3 and §9.3.2.2.)

The lemma is proved by presenting a three-step reduction. The first step is a pre-processing step that makes the underlying graph suitable for further analysis (e.g., the resulting graph will be an expander). The value of vlt may decrease during this step by a constant factor. The heart of the reduction is the second step in which we increase vlt by any desired constant factor. This is done by a construction that corresponds to taking a random walk of constant length on the current graph. The latter step also increases the alphabet Σ, and thus a post-processing step is employed to regain the original alphabet (by using any inner PCP systems; e.g., the one presented in §9.3.2.1). Details follow.

We first stress that the aforementioned Σ and c, as well as the auxiliary parameters d and t (to be introduced in the following two paragraphs), are fixed constants that will be determined such that various conditions (which arise in the course of our argument) are satisfied. Specifically, t will be the last parameter to

[37] For details, see [66].

[38] **Advanced comment:** The augmentation is used to avoid using the Parallel Repetition Theorem of [184]. In the augmented version, with constant probability (say half), a consistency check takes place between tuples that contain copies of the same variable (or query).

be determined (and it will be made greater than a constant that is determined by all the other parameters).

We start with the pre-processing step. Our aim in this step is to reduce the input (G, Φ) of $\mathtt{gapCSP}^{\Sigma}$ to an instance (G_1, Φ_1) such that G_1 is a d-regular expander graph.[39] Furthermore, each vertex in G_1 will have at least $d/2$ self-loops, the number of edges will be preserved up to a constant factor (i.e., $|G_1| = O(|G|)$), and $\mathtt{vlt}(G_1, \Phi_1) = \Theta(\mathtt{vlt}(G, \Phi))$. This step is quite simple: essentially, the original vertices are replaced by expanders of size proportional to their degree, and a big (dummy) expander is superimposed on the resulting graph (see Exercise 9.26).

The main step is aimed at increasing the fraction of violated constraints by a sufficiently large constant factor. The intuition underlying this step is that the probability that a random (t-edge long) walk on the expander G_1 intersects a fixed set of edges is closely related to the probability that a random sample of (t) edges intersects this set. Thus, we may expect such walks to hit a violated edge with probability that is $\min(\Theta(t \cdot \nu), c)$, where ν is the fraction of violated edges. Indeed, the current step consists of reducing the instance (G_1, Φ_1) of $\mathtt{gapCSP}^{\Sigma}$ to an instance (G_2, Φ_2) of $\mathtt{gapCSP}^{\Sigma'}$ such that $\Sigma' = \Sigma^{d^t}$ and the following holds:

1. The vertex set of G_2 is identical to the vertex set of G_1, and each t-edge long path in G_1 is replaced by a corresponding edge in G_2, which is thus a d^t-regular graph.

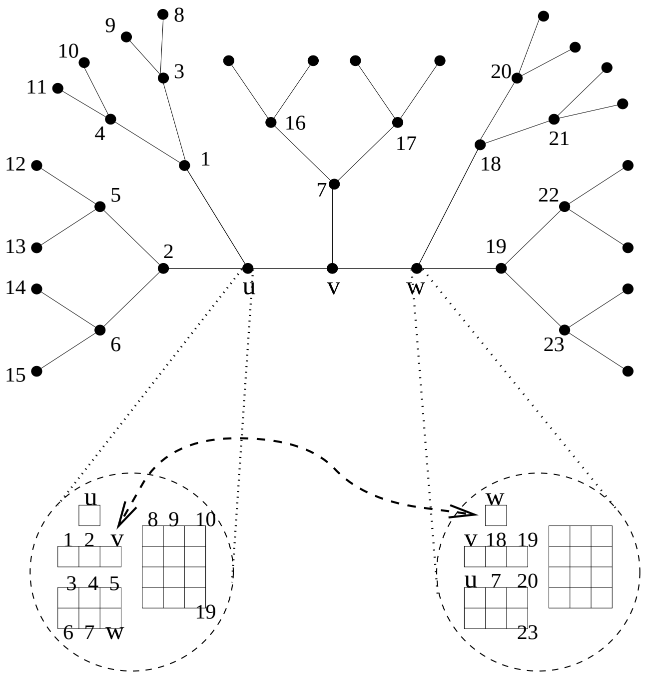
2. The constraints in Φ_2 are the natural ones, viewing each element of Σ' as a Σ-labeling of the ("distance $\leq t$") neighborhood of a vertex (see Figure 9.4), and checking that two such labelings are consistent as well as satisfy Φ_1. That is, the following two types of constraints are introduced:

   (consistency): If there is a path of length at most t in G_1, going from vertex u to vertex w and passing through vertex v, then the Φ_2-constraint associated with the G_2-edge between vertices u and w mandates the equality of the entries corresponding to vertex v in the Σ'-labeling of vertices u and w.

   (satisfying Φ_1): If the G_1-edge (v, v') is on a path of length at most t starting at u then the Φ_2-constraint associated with the G_2-edge that corresponds to this path enforces the Φ_1-constraint that is associated with (v, v').

Clearly, $|G_2| = d^{t-1} \cdot |G_1| = O(|G_1|)$, because d is a constant and t will be set to a constant. (Indeed, the relatively moderate increase in the size of the graph corresponds to the low randomness-complexity of selecting a random walk of length t in G_1.)

[39] A d-regular graph is a graph in which each vertex is incident to exactly d edges. Loosely speaking, an expander graph has the property that each moderately balanced cut (i.e., partition of its vertex set) has relatively many edges crossing it. An equivalent definition, also used in the actual analysis, is that the second eigenvalue of the corresponding adjacency matrix has absolute value that is bounded away from d. For further details, see §E.2.1.1.



The alphabet Σ' as a labeling of the distance $t = 3$ neighborhoods, when repetitions are omitted. In this case $d = 6$ but the self-loops are not shown (and so the "effective" degree is three). The two-sided arrow indicates one of the edges in G_1 that will contribute to the edge constraint between u and w in (G_2, Φ_2).

Figure 9.4: The amplifying reduction in the second proof of the PCP Theorem.

Turning to the analysis of this step, we note that $\mathtt{vlt}(G_1, \Phi_1) = 0$ implies $\mathtt{vlt}(G_2, \Phi_2) = 0$. The interesting fact is that the fraction of violated constraints increases by a factor of $\Omega(\sqrt{t})$; that is, $\mathtt{vlt}(G_2, \Phi_2) \geq \min(\Omega(\sqrt{t} \cdot \mathtt{vlt}(G_1, \Phi_1)), c)$. Here we merely provide a rough intuition and refer the interested reader to [66]. We may focus on any Σ'-labeling to the vertices of G_2 that is consistent with some Σ-labeling of G_1, because relatively few inconsistencies (among the Σ-values assigned to a vertex by the Σ'-labeling of other vertices) can be ignored, while relatively many such inconsistencies yield violation of the "equality constraints" of many edges in G_2. Intuitively, relying on the hypothesis that G_1 is an expander, it follows that the set of violated edge-constraints (of Φ_1) with respect to the aforementioned Σ-labeling causes many more edge-constraints of Φ_2 to be violated (because each edge-constraint of Φ_1 is enforced by many edge-constraints of Φ_2). The point is that *any set F of edges of G_1 is likely to appear on a* $\min(\Omega(t) \cdot |F|/|G_1|, \Omega(1))$ *fraction of the edges of G_2* (i.e., t-paths of G_1). (Note that the claim would have been obvious if G_1 were a complete graph, but it also holds for an expander.)[40]

The factor of $\Omega(\sqrt{t})$ gained in the second step makes up for the constant factor lost in the first step (as well as the constant factor to be lost in the last step). Furthermore, for a suitable choice of the constant t, the aforementioned gain yields an overall constant factor amplification (of $\mathtt{vlt}$). However, so far we obtained an instance of $\mathtt{gapCSP}^{\Sigma'}$ rather than an instance of $\mathtt{gapCSP}^{\Sigma}$, where $\Sigma' = \Sigma^{d^t}$. The purpose of the last step is to reduce the latter instance to an instance of $\mathtt{gapCSP}^{\Sigma}$. This is done by viewing the instance of $\mathtt{gapCSP}^{\Sigma'}$ as a (weak) PCP system (analogously to Exercise 9.25), and composing it with an inner-verifier using the proof composition paradigm outlined in §9.3.2.2. We stress that the inner-verifier used here needs only handle instances of constant size (i.e., having description length $O(d^t \log|\Sigma|)$), and so the verifier presented in §9.3.2.1 will do. The resulting PCP-system uses randomness $r \stackrel{\rm def}{=} \log_2|G_2| + O(d^t \log|\Sigma|)^2$ and a constant number of binary queries, and has rejection probability $\Omega(\mathtt{vlt}(G_2,\Phi_2))$, which is independent of the choice of the constant t. As in Exercise 9.23, for $\Sigma = \{0,1\}^{O(1)}$, we can easily obtain an instance of $\mathtt{gapCSP}^{\Sigma}$, that has a $\Omega(\mathtt{vlt}(G_2,\Phi_2))$ fraction of violated constraints. Furthermore, the size of the resulting instance (which is used as the output (G',Φ') of the three-step reduction) is $O(2^r) = O(|G_2|)$, where the equality uses the fact that d and t are constants. Recalling that $\mathtt{vlt}(G_2,\Phi_2) \geq \min(\Omega(\sqrt{t}\cdot\mathtt{vlt}(G_1,\Phi_1)),c)$ and $\mathtt{vlt}(G_1,\Phi_1) = \Omega(\mathtt{vlt}(G,\Phi))$, this completes the (outline of the) proof of the entire lemma. □

Reflection. In contrast to the proof presented in §9.3.2.2, which combines two remarkable constructs by using a simple composition method, the current proof of the PCP Theorem is based on developing a powerful "combining method" that improves the quality of the main system to which it is applied. This new method, captured by the Amplification Lemma (Lemma 9.19), does not merely obtain the best of the combined systems, but rather obtains a better system than the one given. However, the quality-amplification offered by Lemma 9.19 is rather moderate, and thus many applications are required in order to derive the desired result. Taking the opposite perspective, one may say that remarkable results are obtained by a gradual process of many moderate amplification steps.

9.3.3 PCP and Approximation

The characterization of $\mathcal{NP}$ in terms of probabilistically checkable proofs plays a central role in the study of the complexity of natural approximation problems (cf., Section 10.1.1). To demonstrate this relationship, we first note that any PCP system V gives rise to an approximation problem that consists of estimating the maximum acceptance probability for a given input; that is, on input x, the task is approximating the probability that V accepts x when given oracle access to the best possible π (i.e., we wish to approximate $\max_\pi\{\Pr[V^\pi(x)=1]\}$). Thus, *if $S \in \mathcal{PCP}(r,q)$ then deciding membership in S is reducible to approximating*

[40] We mention that, due to a technical difficulty, it is easier to establish the claimed bound of $\Omega(\sqrt{t}\cdot\mathtt{vlt}(G_1,\Phi_1))$ rather than $\Omega(t\cdot\mathtt{vlt}(G_1,\Phi_1))$.

the maximum among $\exp(2^{r+q})$ *quantities* (corresponding to all effective oracles), where each quantity can be evaluated in time $2^r \cdot \text{poly}$. For (the validity of) this reduction, *an approximation up to a constant factor* (of 2) *will do.*

Note that the foregoing approximation problem is parameterized by a PCP verifier V, and its instances are given their value with respect to this verifier (i.e., the instance x has value $\max_\pi\{\mathsf{Pr}[V^\pi(x)=1]\}$). This per se does not yield a "natural" approximation problem. In order to link PCP systems with natural approximation problems, we take a closer look at the approximation problem associated with $\mathcal{PCP}(r,q)$.

For simplicity, we focus on the case of non-adaptive PCP systems (i.e., all the queries are determined beforehand based on the input and the internal coin tosses of the verifier). Fixing an input x for such a system, we consider the $2^{r(|x|)}$ Boolean formulae that represent the decision of the verifier on each of the possible outcomes of its coin tosses after inspecting the corresponding bits in the proof oracle. That is, each of these $2^{r(|x|)}$ formulae depends on $q(|x|)$ Boolean variables that represent the values of the corresponding bits in the proof oracle. Thus, if x is a yes-instance then there exists a truth assignment (to these variables) that satisfies all $2^{r(|x|)}$ formulae, whereas if x is a no-instance then there exists no truth assignment that satisfies more than $2^{r(|x|)-1}$ formulae. Furthermore, in the case that $r(n) = O(\log n)$, given x, we can construct the corresponding sequence of formulae in polynomial-time. Hence, the PCP Theorem (i.e., Theorem 9.16) yields *NP-hardness results regarding the approximation of the number of simultaneously satisfiable Boolean formulae of constant size.* This motivates the following definition.

Definition 9.20 (gap problems for SAT and generalized-SAT): *For constants* $q \in \mathbb{N}$ *and* $\varepsilon > 0$*, the promise problem* $\mathtt{gapGSAT}_\varepsilon^q$ *refers to instances that are each a sequence of q-variable Boolean formulae* (i.e., each formula depends on at most q variables). *The* yes*-instances are sequences that are simultaneously satisfiable, whereas the* no*-instances are sequences for which no Boolean assignment satisfies more than a* $1-\varepsilon$ *fraction of the formulae in the sequence. The promise problem* $\mathtt{gapSAT}_\varepsilon^q$ *is defined analogously, except that in this case each instance is a sequence of disjunctive clause* (i.e., each formula in each sequence consists of a single disjunctive clause).

Indeed, each instance of $\mathtt{gapSAT}_\varepsilon^q$ is naturally viewed as q-CNF formulae, and we consider an assignment that satisfies as many clauses (of the input CNF) as possible. As hinted, $\mathcal{NP} \subseteq \mathcal{PCP}(\mathtt{log}, O(1))$ implies that $\mathtt{gapGSAT}_{1/2}^{O(1)}$ is NP-complete, which in turn implies that for some constant $\varepsilon > 0$ the problem $\mathtt{gapSAT}_\varepsilon^3$ is NP-complete. The converses hold too. All these claims are stated and proved next.

Theorem 9.21 (equivalent formulations of the PCP Theorem). *The following three conditions are equivalent:*

1. The PCP Theorem: *there exists a constant* q *such that* $\mathcal{NP} \subseteq \mathcal{PCP}(\mathtt{log}, q)$.

2. *There exists a constant* q *such that* $\mathtt{gapGSAT}_{1/2}^q$ *is* $\mathcal{NP}$*-hard.*

3. *There exists a constant* $\varepsilon > 0$ *such that* $\mathtt{gapSAT}^3_\varepsilon$ *is* $\mathcal{NP}$*-hard.*

The point of Theorem 9.21 is not its mere validity (which follows from the validity of each of the three items), but rather the fact that its proof is quite simple. Note that Items 2 and 3 make no reference to PCP. Thus, their (easy to establish) equivalence to Item 1 manifests that the hardness of approximating natural optimization problems lies at the heart of the PCP Theorem. In general, probabilistically checkable proof systems for $\mathcal{NP}$ yield strong inapproximability results for various classical optimization problems (cf., Exercise 9.18 and Section 10.1.1).

Proof: We first show that the PCP Theorem implies the NP-hardness of $\mathtt{gapGSAT}$. We may assume, without loss of generality, that, for some constant q and every $S \in \mathcal{NP}$, it holds that $S \in \mathcal{PCP}(O(\log), q)$ via a non-adaptive verifier (because q adaptive queries can be emulated by 2^q non-adaptive queries). We reduce S to $\mathtt{gapGSAT}$ as follows. On input x, we scan all $2^{O(\log|x|)}$ possible sequence of outcomes of the verifier's coin tosses, and for each such sequence of outcomes we determine the queries made by the verifier as well as the residual decision predicate (where this predicate determines which sequences of answers lead this verifier to accept). That is, for each random-outcome $\omega \in \{0,1\}^{O(\log|x|)}$, we consider the residual predicate, determined by x and ω, that specifies which q-bit long sequence of oracle answers makes the verifier accept x on coins ω. Indeed, this predicate depends only on q variables (which represent the values of the q corresponding oracle answers). Thus, we map x to a sequence of $\mathrm{poly}(|x|)$ formulae, each depending on q variables, obtaining an instance of $\mathtt{gapGSAT}^q$. This mapping can be computed in polynomial-time, and indeed $x \in S$ (resp., $x \notin S$) is mapped to a yes-instance (resp., no-instance) of $\mathtt{gapGSAT}^q_{1/2}$.

Item 2 implies Item 3 by a standard reduction of GSAT to 3SAT. Specifically, $\mathtt{gapGSAT}^q_{1/2}$ reduces to $\mathtt{gapSAT}^q_{2^{-(q+1)}}$, which in turn reduces to $\mathtt{gapSAT}^3_\varepsilon$ for $\varepsilon = 2^{-(q+1)}/(q-2)$. Note that Item 3 implies Item 2 (e.g., given an instance of $\mathtt{gapSAT}^3_\varepsilon$, consider all possible conjunctions of $1/\varepsilon$ disjunctive clauses in the given instance).

We complete the proof by showing that Item 3 implies Item 1. (The same argument shows that Item 2 implies Item 1.) This is done by showing that $\mathtt{gapSAT}^3_\varepsilon$ is in $\mathcal{PCP}(\varepsilon^{-1}\log, 3\varepsilon^{-1})$, and using the reduction of $\mathcal{NP}$ to $\mathtt{gapSAT}^3_\varepsilon$ to derive a corresponding PCP for each set in $\mathcal{NP}$. In fact, we show that $\mathtt{gapGSAT}^q_\varepsilon$ is in $\mathcal{PCP}(\varepsilon^{-1}\log, \varepsilon^{-1}q)$, and do so by presenting a very natural PCP system. In this PCP system the proof oracle is supposed to be an satisfying assignment, and the verifier selects at random one of the (q-variable) formulae in the input sequence, and checks whether it is satisfied by the (assignment given by the) oracle. This amounts to tossing logarithmically many coins and making q queries. This verifier always accepts yes-instances (when given access to an adequate oracle), whereas each no-instances is rejected with probability at least ε (no matter which oracle is used). To amplify the rejection probability (to the desired threshold of 1/2), we invoke the foregoing verifier ε^{-1} times (and note that $(1-\varepsilon)^{1/\varepsilon} < 1/2$). ■

Gap amplifying reductions – a reflection. Item 2 (resp., Item 3) of Theorem 9.21 implies that GSAT (resp., 3SAT) can be reduce to $\mathtt{gapGSAT}_{1/2}$ (resp., to

$\texttt{gapSAT}^3_\varepsilon$). This means that there exist "gap amplifying" reductions of problems like 3SAT to themselves, where these reductions map yes-instances to yes-instances (as usual), while mapping no-instances to no-instances that are "far" from being yes-instances. That is, no-instances are mapped to no-instances of a special type such that a "gap" is created between the yes-instances and no-instances at the image of the reduction. For example, in the case of 3SAT, unsatisfiable formulae are mapped to formulae that are not merely unsatisfiable but rather have no assignment that satisfies more than a $1-\varepsilon$ fraction of the clauses. Thus, PCP constructions are essentially "gap amplifying" reductions.

9.3.4 More on PCP itself: an overview

We start by discussing variants of the PCP characterization of NP, and next turn to PCPs having expressing power beyond NP. Needless to say, the latter systems have super-logarithmic randomness complexity.

9.3.4.1 More on the PCP characterization of NP

Interestingly, the two complexity measures in the PCP-characterization of $\mathcal{NP}$ can be traded off such that at the extremes we get $\mathcal{NP} = \mathcal{PCP}(\texttt{log}, O(1))$ and $\mathcal{NP} = \mathcal{PCP}(0, \texttt{poly})$, respectively.

Proposition 9.22 *For every $S \in \mathcal{NP}$, there exists a logarithmic function ℓ* (i.e., $\ell \in \texttt{log}$) *such that, for every integer function k that satisfies $0 \le k(n) \le \ell(n)$, it holds that $S \in \mathcal{PCP}(\ell - k, O(2^k))$.* (Recall that $\mathcal{PCP}(\texttt{log}, \texttt{poly}) \subseteq \mathcal{NP}$.)

Proof Sketch: By Theorem 9.16, we have $S \in \mathcal{PCP}(\ell, O(1))$. To show that $S \in \mathcal{PCP}(\ell - k, O(2^k))$, we consider an emulation of the corresponding verifier in which we try all possibilities for the $k(n)$-bit long prefix of its random-tape. □

Following the establishment of Theorem 9.16, numerous variants of the PCP Characterization of NP were explored. These variants refer to a finer analysis of various parameters of probabilistically checkable proof systems (for sets in $\mathcal{NP}$). Following is a brief summary of some of these studies.[41]

The length of PCPs. Recall that the effective length of the oracle in any $\mathcal{PCP}(\texttt{log}, \texttt{log})$ system is polynomial (in the length of the input). Furthermore, in the PCP systems underlying the proof of Theorem 9.16 the queries refer only to a polynomially long prefix of the oracle, and so the actual length of these PCPs for $\mathcal{NP}$ is polynomial. Remarkably, *the length of PCPs for $\mathcal{NP}$ can be made nearly-linear* (in the combined length of the input and the standard NP-witness), *while maintaining constant query complexity, where by nearly-linear we mean linear up to a poly-logarithmic factor.* (For details see [35, 66].) This means that a *relatively modest amount of redundancy* in the proof oracle suffices for supporting probabilistic verification via a constant number of probes.

[41] With the exception of works that appeared after [89], we provide no references for the results quoted here. We refer the interested reader to [89, Sec. 2.4.4].

The number of queries in PCPs. Theorem 9.16 asserts that a constant number of queries suffice for PCPs with logarithmic randomness and soundness error of 1/2 (for NP). It is currently known that this constant is at most *five*, whereas with *three* queries one may get arbitrary close to a soundness error of 1/2. The obvious trade-off between the number of queries and the soundness error gives rise to the robust notion of amortized query-complexity, defined as the ratio between the number of queries and (minus) the logarithm (to based 2) of the soundness error. *For every* $\varepsilon > 0$*, any set in* $\mathcal{NP}$ *has a PCP system with logarithmic randomness and amortized query-complexity* $1+\varepsilon$ (cf. [118]), whereas only sets in $\mathcal{P}$ have PCPs of logarithmic randomness and amortized query-complexity less than 1.

Free-bit complexity. The motivation to the notion of free bits came from the PCP–to–MaxClique connection (see Exercise 9.18 and [28, Sec. 8]), but we believe that this notion is of independent interest. Intuitively, this notion distinguishes between queries for which the acceptable answer is determined by previously obtained answers (i.e., the verifier compares the answer to a value determined by the previous answers) and queries for which the verifier only records the answer for future usage. The latter queries are called free (because any answer to them is "acceptable"). For example, in the linearity test (see §9.3.2.1) the first two queries are free and the third is not (i.e., the test accepts if and only if $f(x)+f(y) = f(x+y)$). The amortized free-bit complexity is define analogously to the amortized query complexity. Interestingly, $\mathcal{NP}$ *has PCPs with logarithmic randomness and amortized free-bit complexity less than any positive constant.*

Adaptive versus non-adaptive verifiers. Recall that a PCP verifier is called non-adaptive if its queries are determined solely based on its input and the outcome of its coin tosses. (A general verifier, called adaptive, may determine its queries also based on previously received oracle answers.) Recall that the PCP Characterization of NP (i.e., Theorem 9.16) is established using a non-adaptive verifier; however, it turns out that *adaptive verifiers are more powerful than non-adaptive ones in terms of quantitative results*: Specifically, for PCP verifiers making *three* queries and having logarithmic randomness complexity, adaptive queries provide for soundness error at most 0.51 (actually $0.5+\varepsilon$ for any $\varepsilon > 0$) for any set in $\mathcal{NP}$, whereas *non-adaptive* queries provide soundness error 5/8 (or less) only for sets in $\mathcal{P}$.

Non-binary queries. Our definition of PCP allows only binary queries. Certainly, non-binary queries can be emulated by binary queries, but the converse does not necessarily hold.[42] For this reason, "parallel repetition" is highly non-trivial

[42]**Advanced comment:** The source of trouble is the adversarial settings (implicit in the soundness condition), which means that when several binary queries are packed into one non-binary query, the adversary need not respect the packing (i.e., it may answer inconsistently on the same binary query depending on the other queries packed with it). This trouble becomes acute in the case of PCPs, because they do not correspond to a full information game. Indeed, in contrast, parallel repetition is easy to analyze in the case of interactive proof systems, because they can be modeled as full information games: this is obvious in the case of public-coin systems, but also holds for general interactive proof systems (see Exercise 9.1).

in the PCP setting. Still, a Parallel Repetition Theorem that refers to independent invocations of the same PCP is known, but it is not applicable for obtaining soundness error smaller than a constant (while preserving logarithmic randomness). Nevertheless, using adequate "consistency tests" one may construct PCP systems for $\mathcal{NP}$ using logarithmic randomness, a constant number of (non-binary) queries and *soundness error exponential in the length of the answers.* (Currently, this is known only for sub-logarithmic answer lengths.)

9.3.4.2 Stronger forms of PCP systems for NP

Although the PCP Theorem is famous mainly for its negative applications to the study of natural approximation problems (see Section 9.3.3 and §10.1.1.2), its potential for direct positive applications is fascinating. Indeed, the vision of speeding-up the verification of mundane proofs is exciting, where these proofs may refer to mundane assertions such as the correctness of a specific computation. Enabling such a speed-up requires a strengthening of the PCP Theorem such that it mandates efficient verification time rather than "merely" low query-complexity of the verification task. Such a strengthening is possible.

Theorem 9.23 (Theorem 9.16 – strengthened): *Every set S in $\mathcal{NP}$ has a PCP system V of logarithmic randomness-complexity, constant query-complexity, and quadratic time-complexity. Furthermore, NP-witnesses for membership in S can be transformed in polynomial-time to corresponding proof-oracles for V.*

The furthermore part was already stated in Section 9.3.2 (as a strengthening of Theorem 9.16). Thus, the novelty in Theorem 9.23 is that it provides quadratic verification time, rather than polynomial verification time (where the polynomial may depend arbitrarily on the set S). Theorem 9.23 is proved by noting that that the CNF formulae that is obtained by reducing S to 3SAT are highly uniform, and thus the verifier V that is outlined in §9.3.2.2 can be implemented in quadratic time. Indeed, the most time-consuming operation required of V is evaluating the low-degree extension Φ (of C_ϕ), which corresponds to the input formula ϕ, at a few points. In the context of §9.3.2.2, evaluating Φ in exponential-time suffices (since this means time that is polynomial in $|\phi|$). Theorem 9.23 follows by showing that a variant of Φ can be evaluated in polynomial-time (since this means time that is polylogarithmic in $|\phi|$); for details, see Exercise 9.29.

PCPs of Proximity. Clearly, we cannot expect a PCP system (or any standard proof system for that matter) to have sub-linear verification time (since linear-time is required for merely reading the input). Nevertheless, we may consider a relaxation of the verification task (regarding proofs of membership in a set S). In this relaxation the verifier is only required to reject any input that is "far" from S (regardless of the alleged proof), and, as usual, accept any input that is in S (when accompanied with an adequate proof). Specifically, in order to allow sub-linear time verification, we provide the verifier V with direct access to the bits of the input (which is viewed as an oracle) as well as with direct access to the usual (PCP) proof-oracle, and require that the following two conditions hold (with respect to some constant $\varepsilon > 0$):

Completeness: For every $x \in S$ there exists a string π_x such that, when given access to the oracles x and π_x, machine V always accepts.

Soundness with respect to proximity ε: For every string x that is ε-far from S (i.e., for every $x' \in \{0,1\}^{|x|} \cap S$ it holds that x and x' differ on at least $\varepsilon|x|$ bits) and every string π, when given access to the oracles x and π, machine V rejects with probability at least $\frac{1}{2}$.

Machine V is called a PCP of proximity, and its queries to both oracles are counted in its query-complexity. (Indeed, such a PCP of proximity was used in §9.3.2.2, and the notion is analogous to a relaxation of decision problems that is reviewed in Section 10.1.2.)

We mention that *every set in $\mathcal{NP}$ has a PCPs of proximity of logarithmic randomness-complexity, constant query-complexity, and polylogarithmic time-complexity.* This follows by using ideas as underlying the proof of Theorem 9.23 (see also Exercise 9.29).

9.3.4.3 PCP with super-logarithmic randomness

Our focus so far was on the important case where the verifier tosses logarithmically many coins, and hence the "effective proof length" is polynomial. Here we mention that the PCP Theorem (or rather Theorem 9.23) scales up.[43]

Theorem 9.24 (Theorem 9.16 – Generalized): *Let $t(\cdot)$ be an integer function such that $n<t(n)<2^{\mathrm{poly}(n)}$. Then,* $\textsc{Ntime}(t) \subseteq \mathcal{PCP}(O(\log t), O(1))$.

Recall that $\mathcal{PCP}(r,q) \subseteq \textsc{Ntime}(t)$, for $t(n) = \mathrm{poly}(n) \cdot 2^{r(n)}$. Thus, the Ntime Hierarchy implies a hierarchy of $\mathcal{PCP}(\cdot, O(1))$ classes, for randomness complexity ranging between logarithmic and polynomial functions.

Chapter Notes

(The following historical notes are quite long and still they fail to properly discuss several important technical contributions that played an important role in the development of the area. For further details, the reader is referred to [89, Sec. 2.6.2].)

Motivated by the desire to formulate the most general type of "proofs" that may be used within cryptographic protocols, Goldwasser, Micali and Rackoff [108] introduced the notion of an *interactive proof system.* Although the main thrust of their work was the introduction of a special type of interactive proofs (i.e., ones that are *zero-knowledge*), the possibility that interactive proof systems may be more powerful from NP-proof systems was pointed out in [108]. Independently of [108],

[43] Note that the sketched proof of Theorem 9.23 yields verification time that is quadratic in the length of the input and polylogarithmic in the length of the NP-witness.

Babai [17] suggested a different formulation of interactive proofs, which he called *Arthur-Merlin Games*. Syntactically, Arthur-Merlin Games are a restricted form of interactive proof systems, yet it was subsequently shown that these restricted systems are as powerful as the general ones (cf., [110]). The speed-up result (i.e., $\mathcal{AM}(2f) \subseteq \mathcal{AM}(f)$) is due to [22] (improving over [17]).

The first evidence to the power of interactive proofs was given by Goldreich, Micali, and Wigderson [99], who presented an interactive proof system for Graph Non-Isomorphism (Construction 9.3). More importantly, they demonstrated the *generality and wide applicability of zero-knowledge proofs*: Assuming the existence of one-way function, they showed how to construct zero-knowledge interactive proofs for any set in $\mathcal{NP}$ (Theorem 9.11). This result has had a dramatic impact on the design of cryptographic protocols (cf., [100]). For further discussion of zero-knowledge and its applications to cryptography, see Appendix C. Theorem 9.12 (i.e., $\mathcal{ZK} = \mathcal{IP}$) is due to [31, 129].

Probabilistically checkable proof (PCP) systems are related to *multi-prover interactive proof systems*, a generalization of interactive proofs that was suggested by Ben-Or, Goldwasser, Kilian and Wigderson [32]. Again, the main motivation came from the zero-knowledge perspective; specifically, presenting multi-prover zero-knowledge proofs for $\mathcal{NP}$ without relying on intractability assumptions. Yet, the complexity theoretic prospects of the new class, denoted $\mathcal{MIP}$, have not been ignored.

The amazing power of interactive proof systems was demonstrated by using algebraic methods. The basic technique was introduced by Lund, Fortnow, Karloff and Nisan [161], who applied it to show that the polynomial-time hierarchy (and actually $\mathcal{P}^{\#\mathcal{P}}$) is in $\mathcal{IP}$. Subsequently, Shamir [204] used the technique to show that $\mathcal{IP} = \mathcal{PSPACE}$, and Babai, Fortnow and Lund [19] used it to show that $\mathcal{MIP} = \mathcal{NEXP}$. (Our entire proof of Theorem 9.4 follows [204].)

The aforementioned multi-prover proof system of Babai, Fortnow and Lund [19] (hereafter referred to as the BFL proof system) has been the starting point for fundamental developments regarding $\mathcal{NP}$. The first development was the discovery that the BFL proof system can be "scaled-down" from $\mathcal{NEXP}$ to $\mathcal{NP}$. This important discovery was made independently by two sets of authors: Babai, Fortnow, Levin, and Szegedy [20] and Feige, Goldwasser, Lovász, and Safra [72]. However, the manner in which the BFL proof is scaled-down is different in the two papers, and so are the consequences of the scaling-down.

Babai *et. al.* [20] start by considering (only) inputs encoded using a special error-correcting code. The encoding of strings, relative to this error-correcting code, can be computed in polynomial time. They presented an almost-linear time algorithm that transforms NP-witnesses (to inputs in a set $S \in \mathcal{NP}$) into *transparent proofs* that can be verified (as vouching for the correctness of the encoded assertion) in (probabilistic) *poly-logarithmic time* (by a Random Access Machine). Babai *et. al.* [20] stress the practical aspects of transparent proofs; specifically, for rapidly checking transcripts of long computations.

In contrast, in the proof system of Feige *et. al.* [72, 73] the verifier stays polynomial-time and only two more refined complexity measures (i.e., the randomness and query complexities) are reduced to poly-logarithmic. This eliminates the need to assume that the input is in a special error-correcting form, and yields a refined (quantitative) version of the notion of probabilistically checkable proof systems (introduced in [79]), where the refinement is obtained by specifying the randomness and query complexities (see Definition 9.14). Hence, whereas the BFL proof system [19] can be reinterpreted as establishing $\mathcal{NEXP} = \mathcal{PCP}(\mathtt{poly}, \mathtt{poly})$, the work of Feige *et. al.* [73] establishes $\mathcal{NP} \subseteq \mathcal{PCP}(f, f)$, where $f(n) = O(\log n \cdot \log\log n)$. (In retrospect, we note that the work of Babai *et. al.* [20] implies that $\mathcal{NP} \subseteq \mathcal{PCP}(\mathtt{log}, \mathtt{polylog})$.)

Interest in the new complexity class became immense since Feige *et. al.* [72, 73] demonstrated its relevance to proving the intractability of approximating some natural combinatorial problems (specifically, for MaxClique). When using the PCP–to–MaxClique connection established by Feige *et. al.*, the randomness and query complexities of the verifier (in a PCP system for an NP-complete set) relate to the strength of the negative results obtained for the approximation problems. This fact provided a very strong motivation for trying to reduce these complexities and obtain a tight characterization of $\mathcal{NP}$ in terms of $\mathcal{PCP}(\cdot, \cdot)$. The obvious challenge was showing that $\mathcal{NP}$ equals $\mathcal{PCP}(\mathtt{log}, \mathtt{log})$. This challenge was met by Arora and Safra [15]. Actually, they showed that $\mathcal{NP} = \mathcal{PCP}(\mathtt{log}, q)$, where $q(n) = o(\log n)$.

Hence, a new challenge arose; namely, further reducing the query complexity – in particular, to a constant – while maintaining the logarithmic randomness complexity. Again, additional motivation for this challenge came from the relevance of such a result to the study of natural approximation problems. The new challenge was met by Arora, Lund, Motwani, Sudan and Szegedy [14], and is captured by the PCP Characterization Theorem, which asserts that $\mathcal{NP} = \mathcal{PCP}(\mathtt{log}, O(1))$.

Indeed the PCP Characterization Theorem is a culmination of a sequence of impressive works [161, 19, 20, 73, 15, 14]. These works are rich in innovative ideas (e.g., various arithmetizations of SAT as well as various forms of proof composition) and employ numerous techniques (e.g., low-degree tests, self-correction, and pseudorandomness). Our overview of the original proof of the PCP Theorem (in §9.3.2.1–9.3.2.2) is based on [14, 15].[44] The alternative proof outlined in §9.3.2.3 is due to Dinur [66].

We mention some of the ideas and techniques involved in deriving even stronger variants of the PCP Theorem (which are surveyed in §9.3.4.1). These include the Parallel Repetition Theorem [184], the use of the Long-Code [28], and the application of Fourier analysis in this setting [115, 116]. We also highlight the notions of PCPs of proximity and robustness (see [34, 67]).

Computationally-Sound Proof Systems. Argument systems were defined by Brassard, Chaum and Crépeau [48], with the motivation of providing *perfect* zero-knowledge arguments (rather than zero-knowledge *proofs*) for $\mathcal{NP}$. A few years later, Kilian [144] demonstrated their significance beyond the domain of zero-knowledge by showing that, under some reasonable intractability assumptions, ev-

[44] Our presentation also benefits from the notions of PCPs of proximity and robustness, put forward in [34, 67].

ery set in $\mathcal{NP}$ has a computationally-sound proof in which the randomness and communication complexities are poly-logarithmic.[45] Interestingly, these argument systems rely on the fact that $\mathcal{NP} \subseteq \mathcal{PCP}(f, f)$, for $f(n) = \text{poly}(\log n)$. We mention that Micali [164] suggested a different type of computationally-sound proof systems (which he called CS-proofs).

Final comment: The current chapter is a revision of [89, Chap. 2]. In particular, more details are provided here for the main topics, whereas numerous secondary topics discussed in [89, Chap. 2] are not mentioned here (or are only briefly mentioned here). We note that a few of the research directions that were mentioned in [89, Sec. 2.4.4] have received considerable attention in the period that elapsed, and improved results are currently known. In particular, the interested reader is referred to [34, 35, 66] for a study of the length of PCPs, and to [118] for a study of their amortized query complexity. Likewise, a few open problems mentioned in [89, Sec. 2.6.3] have been resolved; specifically, the interested reader is referred to [24, 171] for breakthrough results regarding zero-knowledge.

Exercises

Exercise 9.1 (parallel error-reduction for interactive proof systems) Prove that the error probability (in the soundness condition) can be reduced by parallel repetitions of the proof system. (A proof appears in [89, Apdx. C.1].)

Guideline: As a warm-up, consider the special case of public-coin interactive proof systems. Next, generalize the analysis to arbitrary interactive proof systems, by considering (as a mental experiment) a "powerful verifier" that emulates the original verifier while behaving as in the public-coin model.

Exercise 9.2 Prove that if S is Karp-reducible to a set in $\mathcal{IP}$, then $S \in \mathcal{IP}$. Prove that if S is Cook-reducible to a set S' such that both S' and $\{0,1\}^* \setminus S'$ are in $\mathcal{IP}$, then $S \in \mathcal{IP}$.

Exercise 9.3 Complete the details of the proof that co$\mathcal{NP} \subseteq \mathcal{IP}$ (i.e., the first part of the proof of Theorem 9.4). In particular, suppose that the protocol for unsatisfiability is applied to a CNF formula with n variables and m clauses. Then, what is the length of the messages sent by the two parties? What is the soundness error?

Exercise 9.4 Present an interactive proof system for unsatisfiability such that on input a CNF formula having n variables the parties exchange $n/O(\log n)$ messages.

Guideline: Modify the (first part of the) proof of Theorem 9.4, by stripping $O(\log n)$ summations in each round.

[45] We comment that interactive proofs are unlikely to have such low complexities; see [105].

Exercise 9.5 (an interactive proof system for $\#\mathcal{P}$) Using the main part of the proof of Theorem 9.4, present a proof system for the counting set of Eq. (9.5).

Guideline: Use a slightly different arithmetization of CNF formulae. Specifically, instead of replacing the clause $x \vee \neg y \vee z$ by the term $(x+(1-y)+z)$, replace it by the term $(1-((1-x)\cdot y\cdot(1-z)))$. The point is that this arithmetization maps Boolean assignments that satisfy the CNF formula to 0-1 assignments that when substituted in the corresponding arithmetic expression yield the value 1 (rather than yielding a somewhat arbitrary positive integer).

Exercise 9.6 Show that QBF can be reduced to a special form of (non-canonical)[46] QBF in which no variable appears both to the left and to the right of more than one universal quantifier.

Guideline: Consider a process (which proceeds from left to right) of "refreshing" variables after each universal quantifier. Let $\phi(x_1,...,x_s,y,x_{s+1},...,x_{s+t})$ be a quantifier-free boolean formula and let $Q_{s+1},...,Q_{s+t}$ be an arbitrary sequence of quantifiers. Then, we replace the quantified (sub-)formula

$$\forall y Q_{s+1}x_{s+1} \cdots Q_{s+t}x_{s+t}\,\phi(x_1,...,x_s,y,x_{s+1},...,x_{s+t})$$

by the (sub-)formula

$$\forall y \exists x'_1 \cdots \exists x'_s [(\wedge_{i=1}^s (x'_i = x_i)) \ \wedge\ Q_{s+1}x_{s+1} \cdots Q_{s+t}x_{s+t}\,\phi(x'_1,...,x'_s,y,x_{s+1},...,x_{s+t})]\,.$$

Note that the variables $x_1,...,x_s$ do not appear to the right of the quantifier Q_{s+1} in the replaced formula, and that the length of the replaced formula grows by an additive term of $O(s)$. This process of refreshing variables is applied from left to right on the entire sequence of universal quantifiers (except the inner one, for which this refreshing is useless).[47]

Exercise 9.7 Prove that if two integers in $[0,M]$ are different then they must be different modulo most of the primes in the interval $[3,L]$, where $L=\mathrm{poly}(\log M)$]. Prove the same for the interval $[L,2L]$.

Guideline: Let $a \neq b \in [0,M]$ and suppose that $P_1,...,P_t$ is an enumeration of all the primes that satisfy $a \equiv b \pmod{P_i}$. Using the Chinese Reminder Theorem, prove that $Q \stackrel{\rm def}{=} \prod_{i=1}^t P_i \leq M$ (because otherwise $a=b$ follows by combining $a \equiv b \pmod{Q}$ with the hypothesis $a,b \in [0,M]$). It follows that $t < \log_2 M$. Using a lower-bound on the density of prime numbers, the claim follows.

[46] See Appendix G.2.

[47] For example,

$$\exists z_1 \forall z_2 \exists z_3 \forall z_4 \exists z_5 \forall z_6\, \phi(z_1,z_2,z_3,z_4,z_5,z_6)$$

is first replaced by

$$\exists z_1 \forall z_2 \exists z'_1\, [(z'_1 = z_1) \wedge \exists z_3 \forall z_4 \exists z_5 \forall z_6\, \phi(z'_1,z_2,z_3,z_4,z_5,z_6)]$$

and next (written as $\exists z_1 \forall z'_2 \exists z'_1\, [(z'_1 = z_1) \wedge \exists z'_3 \forall z'_4 \exists z'_5 \forall z'_6\, \phi(z'_1,z'_2,z'_3,z'_4,z'_5,z'_6)])$ is replaced by

$$\exists z_1 \forall z'_2 \exists z'_1\, [(z'_1 = z_1) \wedge \exists z'_3 \forall z'_4 \exists z''_1 \exists z''_2 \exists z''_3$$
$$[(\wedge_{i=1}^3 (z''_i = z'_i)) \wedge \exists z'_5 \forall z'_6 \phi(z''_1,z''_2,z''_3,z'_4,z'_5,z'_6)]].$$

Thus, in the resulting formula, no variable appears both to the left and to the right of more than a single universal quantifier.

Exercise 9.8 (on interactive proofs with two-sided error (following [81])) Let $\mathcal{IP}'(f)$ denote the class of sets having a two-sided error interactive proof system in which a total of $f(|x|)$ messages are exchanged on common input x. Specifically, suppose that a suitable prover may cause every yes-instance to be accepted with probability at least $2/3$ (rather than 1), while no cheating prover can cause a no-instance to be accepted with probability greater than $1/3$ (rather than $1/2$). Similarly, let $\mathcal{AM}'$ denote the public-coin version of $\mathcal{IP}'$.

1. Establish $\mathcal{IP}'(f) \subseteq \mathcal{AM}'(f+3)$ by noting that the proof of Theorem F.2, which establishes $\mathcal{IP}(f) \subseteq \mathcal{AM}(f+3)$, extends to the two-sided error setting.
2. Prove that $\mathcal{AM}'(f) \subseteq \mathcal{AM}(f+1)$ by extending the ideas underlying the proof of Theorem 6.9, which actually establishes that $\mathcal{BPP} \subseteq \mathcal{AM}(1)$ (where $\mathcal{BPP} = \mathcal{AM}'(0)$).

Using the Round Speed-up Theorem (i.e., Theorem F.3), conclude that, for every function $f : \mathbb{N} \to \mathbb{N} \setminus \{1\}$, it holds that $\mathcal{IP}'(f) = \mathcal{AM}(f) = \mathcal{IP}(f)$.

Guideline (for Part 2): Fixing an optimal prover strategy for the given two-sided error public-coin interactive proof, consider the set of verifier coins that make the verifier accept any fixed yes-instance, and apply the ideas underlying the transformation of $\mathcal{BPP}$ to $\mathcal{MA} = \mathcal{AM}(1)$. For further details, see [81].

Exercise 9.9 In continuation to Exercise 9.8, show that $\mathcal{IP}'(f) = \mathcal{IP}(f)$ for every function $f : \mathbb{N} \to \mathbb{N}$ (including $f \equiv 1$).

Guideline: Focus on establishing $\mathcal{IP}'(1) = \mathcal{IP}(1)$, which is identical to Part 2 of Exercise 6.12. Note that the relevant classes defined in Exercise 6.12 coincide with $\mathcal{IP}(1)$ and $\mathcal{IP}'(1)$; that is, $\mathcal{MA} = \mathcal{IP}(1)$ and $\mathcal{MA}^{(2)} = \mathcal{IP}'(1)$.

Exercise 9.10 Prove that every $\mathcal{PSPACE}$-complete set S has an interactive proof system in which the designated prover can be implemented by a probabilistic polynomial-time oracle machine that is given oracle access to S.

Guideline: Use Theorem 9.4 and Proposition 9.5.

Exercise 9.11 (checkers (following [38])) A probabilistic polynomial-time oracle machine C is called a checker for the decision problem Π if the following two conditions hold:

1. For every x it holds that $\Pr[C^{\Pi}(x)=1] = 1$, where (as usual) $C^f(x)$ denotes the output of A on input x when given oracle access to f.
2. For every $f : \{0,1\}^* \to \{0,1\}$ and every x such that $f(x) \neq \Pi(x)$ it holds that $\Pr[C^f(x)=1] \leq 1/2$.

Note that nothing is required in the case that $f(x) = \Pi(x)$ but $f \neq \Pi$. Prove that *if both $S_1 = \{x : \Pi(x)=1\}$ and $S_0 = \{x : \Pi(x)=0\}$ have interactive proof systems in which the designated prover can be implemented by a probabilistic polynomial-time oracle machine that is given oracle access to Π, then Π has a checker.* Using Exercise 9.10, conclude that any $\mathcal{PSPACE}$-complete problem has a checker.

Guideline: On input x and oracle access to f, the checker first obtains $\sigma \stackrel{\rm def}{=} f(x)$. The claim $\Pi(x) = \sigma$ is then checked by combining the verifier of S_σ with the probabilistic polynomial-time oracle machine that describes the designated prover, while referring its queries to the oracle f.

Exercise 9.12 (weakly optimal deciders for checkable problems (following [132])) Prove that if a decision problem Π has a checker (as defined in Exercise 9.11) then there exists a probabilistic algorithm A that satisfies the following two conditions:

1. A solves the decision problem Π (i.e., for every x it holds that $\mathsf{Pr}[A(x) = \Pi(x)] \geq 2/3$).
2. For every probabilistic algorithm A' that solves the decision problem Π, there exists a polynomial p such that for every x it holds that $t_A(x) = p(|x|) \cdot \max_{|x'| \leq p(|x|)}\{t_{A'}(x')\}$, where $t_A(z)$ (resp., $t_{A'}(z)$) denotes the number of steps taken by A (resp., A') on input z.

Note that, compared to Theorem 2.33, the claim of optimality is weaker, but on the other hand it applies to decision problems (rather than to candid search problems).

Guideline: Use the ideas of the proof of Theorem 2.33, noting that the correctness of the answers provided by the various candidate algorithms can be verified by using the checker. That is, A invokes copies of the checker, while using different candidate algorithms as oracles in the various copies.

Exercise 9.13 (on the role of soundness error in zero-knowledge proofs) Prove that if S has a zero-knowledge interactive proof system with perfect soundness (i.e., the soundness error equals zero) then $S \in \mathcal{BPP}$.

Guideline: Let M be an arbitrary algorithm that simulates the view of the (honest) verifier. Consider the algorithm that on input x, accepts x if and only if $M(x)$ represents a valid view of the verifier in an accepting interaction (i.e., an interaction that leads the verifier to accept the common input x). Use the simulation condition to analyze the case $x \in S$, and the perfect soundness hypothesis to analyze the case $x \notin S$.

Exercise 9.14 (on the role of interaction in zero-knowledge proofs) Prove that if S has a zero-knowledge interactive proof system with a uni-directional communication then $S \in \mathcal{BPP}$.

Guideline: Let M be an arbitrary algorithm that simulates the view of the (honest) verifier, and let $M'(x)$ denote the part of this view that consists of the prover message. Consider the algorithm that on input x, obtains $m \leftarrow M'(x)$, and emulates the verifier's decision on input x and message m. Note that this algorithm ignores the part of $M(x)$ that represents the verifier's internal coin tosses, and uses fresh verifier's coins when deciding on (x, m).

Exercise 9.15 (on the effective length of PCP oracles) Suppose that V is a PCP verifier of query-complexity q and randomness-complexity r. Show that for every fixed x, the number of possible locations in the proof oracle that are examined by V on input x (when considering all possible internal coin tosses of V and all possible answers it may receive) is upper-bounded by $2^{q(|x|)+r(|x|)}$. Show that if V is non-adaptive then the upper-bound can be improved to $2^{r(|x|)} \cdot q(|x|)$.

Guideline: In the non-adaptive case, all q queries are determined by V's internal coin tosses.

Exercise 9.16 (on the effective randomness of PCPs) Suppose that a set S has a PCP of query-complexity q that utilizes proof oracles of length ℓ. Show that, for every constant $\varepsilon > 0$, the set S has a "non-uniform" PCP of query complexity q, soundness error $0.5 + \varepsilon$ and randomness complexity r such that $r(n) = \log_2(\ell(n) + n) + O(1)$. By a "non-uniform PCP" we mean one in which the verifier is a probabilistic polynomial-time oracle machine that is given direct access to the bits of a non-uniform $\text{poly}(\ell(n) + n)$-bit long advice.

Guideline: Consider a PCP verifier V as in the hypothesis, and denote its randomness complexity by r_V. We construct a non-uniform verifier V' that, on input of length n, obtains as advice a set $R_n \subseteq \{0,1\}^{r_V(n)}$ of cardinality $O((\ell(n) + n)/\varepsilon^2)$, and emulates V on a uniformly selected element of R_n. Show that for a random R_n of the said size, the verifier V' satisfies the claims of the exercise.
(Extra hint: Fixing any input $x \notin S$ and any oracle $\pi \in \{0,1\}^{\ell(|x|)}$, upper-bound the probability that a random set R_n (of the said size) is bad, where R_n is bad if V accept x with probability $0.5 + \varepsilon$ when selecting its coins in R_n and using the oracle π.)

Exercise 9.17 (on the complexity of sets having certain PCPs) Suppose that a set S has a PCP of query-complexity q and randomness-complexity r. Show that S can be decided by a non-deterministic machine[48] that, on input of length n, makes at most $2^{r(n)} \cdot q(n)$ truly non-deterministic steps (i.e., choosing between different alternatives) and halts within a total number of $2^{r(n)} \cdot \text{poly}(n)$ steps. Conclude that $S \in \text{NTIME}(2^r \cdot \text{poly}) \cap \text{DTIME}(2^{2^r q + r} \cdot \text{poly})$.

Guideline: For each input $x \in S$ and each possible value $\omega \in \{0,1\}^{r(|x|)}$ of the verifier's random-tape, we consider a sequence of $q(|x|)$ bit values that represent a sequence of oracle answers that make the verifier accept. Indeed, for fixed x and $\omega \in \{0,1\}^{r(|x|)}$, each setting of the $q(|x|)$ oracle answers determine the computation of the corresponding verifier (including the queries it makes).

Exercise 9.18 (The FGLSS-reduction [73]) For any $S \in \mathcal{PCP}(r, q)$, consider the following mapping of instances for S to instances of the `Independent Set` problem. The instance x is mapped to a graph $G_x = (V_x, E_x)$, where $V_x \subseteq \{0,1\}^{r(|x|)+q(|x|)}$ consists of pairs (ω, α) such that the PCP verifier *accepts* the input x, when using coins $\omega \in \{0,1\}^{r(|x|)}$ and receiving the answers $\alpha = \alpha_1 \cdots \alpha_{q(|x|)}$ (to the oracle queries determined by x, r and the previous answers). Note that V_x contains only *accepting* "views" of the verifier. The set E_x consists of edges that connect vertices that represents mutually *inconsistent* views of the said verifier; that is, the vertex $v = (\omega, \alpha_1 \cdots \alpha_{q(|x|)})$ is connected to the vertex $v' = (\omega', \alpha'_1 \cdots \alpha'_{q(|x|)})$ if there exists i and i' such that $\alpha_i \neq \alpha'_{i'}$ and $\mathtt{q}_i^x(v) = \mathtt{q}_{i'}^x(v')$, where $\mathtt{q}_i^x(v)$ (resp.,

[48] See §4.2.1.3 for definition of non-deterministic machines.

$\mathsf{q}^x_{i'}(v')$) denotes the i-th (resp., i'-th) query of the verifier on input x, when using coins ω (resp., ω') and receiving the answers $\alpha_1 \cdots \alpha_{i-1}$ (resp., $\alpha'_1 \cdots \alpha'_{i'-1}$). In particular, for every $\omega \in \{0,1\}^{r(|x|)}$ and $\alpha \neq \alpha'$, if $(\omega,\alpha),(\omega,\alpha') \in V_x$, then $\{(\omega,\alpha),(\omega,\alpha')\} \in E_x$.

1. Prove that the mapping $x \mapsto G_x$ can be computed in time that is polynomial in $2^{r(|x|)+q(|x|)} \cdot |x|$.

   (Note that the number of vertices in G_x is upper-bounded by $2^{r(|x|)+f(|x|)}$, where $f \leq q$ is the free-bit complexity of the PCP verifier.)

2. Prove that, for every x, the size of the maximum independent set in G_x is at most $2^{r(|x|)}$.

3. Prove that if $x \in S$ then G_x has an independent set of size $2^{r(|x|)}$.

4. Prove that if $x \notin S$ then the size of the maximum independent set in G_x is at most $2^{r(|x|)-1}$.

In general, denoting the PCP verifier by V, prove that the size of the maximum independent set in G_x is exactly $2^{r(|x|)} \cdot \max_\pi\{\mathsf{Pr}[V^\pi(x)=1]\}$. (Note the similarity to the proof of Proposition 2.26.)
Show that the PCP Theorem implies that *the size of the maximum independent set* (resp., clique) in a graph *is NP-hard to approximate to within any constant factor.*

Guideline: Note that an independent set in G_x corresponds to a set of coins R and a partial oracle π' such that V accepts x when using coins in R and accessing any oracle that is consistent with π'. The FGLSS-reduction creates a gap of a factor of 2 between yes- and no-instances of S (having a standard PCP). Larger factors can be obtained by considering a PCP that results from repeating the original PCP for a constant number of times. The result for `Clique` follows by considering the complement graph.

Exercise 9.19 Using the ideas of Exercise 9.18, prove that, for any $t(n) = o(\log n)$, it holds that $\mathcal{NP} \subseteq \mathcal{PCP}(t,t)$ implies that $\mathcal{P} = \mathcal{NP}$.

Guideline: We only use the fact that the FGLSS-reduction maps instances of $S \in \mathcal{PCP}(t,t)$ to instances of the `Clique` problem (and ignore the fact that we actually get a stronger reduction to a "gap-Clique" problem). Furthermore, when applies to problems in $\mathcal{NP} \subseteq \mathcal{PCP}(t,t)$, the FGLSS-reduction runs in polynomial-time. The key observation is that the FGLSS-reduction maps instances of the `Clique` problem (which is in $\mathcal{NP} \subseteq \mathcal{PCP}(o(\log), o(\log))$) to shorter instances of the same problem (because $2^{o(\log n)} \ll n$). Thus, iteratively applying the FGLSS-reduction, we can reduce instances of `Clique` to instances of constant size. This yields a reduction of `Clique` to a finite set, and $\mathcal{NP} = \mathcal{P}$ follows (by the $\mathcal{NP}$-completeness of `Clique`).

Exercise 9.20 (a simple but partial analysis of the BLR Linearity Test) For Abelian groups G and H, consider functions from G to H. For such a (generic) function f, consider the linearity (or rather homomorphism) test that selects uniformly $r, s \in G$ and checks that $f(r)+f(s) = f(r+s)$. Let $\delta(f)$ denote the distance of f from the set of homomorphisms (of G to H); that is, $\delta(f)$ is the minimum taken over all homomorphisms $h : G \to H$ of $\mathsf{Pr}_{x\in G}[f(x) \neq h(x)]$. Using the following guidelines, prove that the probability that the test rejects f, denoted $\varepsilon(f)$, is at least $3\delta(f) - 6\delta(f)^2$.

1. Suppose that h is the homomorphism closest to f (i.e., $\delta(f) = \mathsf{Pr}_{x\in G}[f(x) \neq h(x)]$). Prove that $\varepsilon(f) = \mathsf{Pr}_{x,y\in G}[f(x) + f(y) \neq f(x+y)]$ is lower-bounded by $3 \cdot \mathsf{Pr}_{x,y}[f(x) \neq h(x) \wedge f(y) = h(y) \wedge f(x+y) = h(x+y)]$.

   (Hint: consider three out of four *disjoint* cases (regarding $f(x) \stackrel{?}{=} h(x)$, $f(y) \stackrel{?}{=} h(y)$, and $f(x+y) \stackrel{?}{=} h(x+y)$) that are possible when $f(x)+f(y) \neq f(x+y)$, where these three cases refer to the disagreement of h and f on exactly one out of the three relevant points.)

2. Prove that $\mathsf{Pr}_{x,y}[f(x) \neq h(x) \wedge f(y) = h(y) \wedge f(x+y) = h(x+y)] \geq \delta(f) - 2\delta(f)^2$.

   (Hint: lower-bound the said probability by $\mathsf{Pr}_{x,y}[f(x) \neq h(x)] - (\mathsf{Pr}_{x,y}[f(x) \neq h(x) \wedge f(y) \neq h(y)] + \mathsf{Pr}_{x,y}[f(x) \neq h(x) \wedge f(x+y) \neq h(x+y)])$.)

Note that the lower-bound $\varepsilon(f) \geq 3\delta(f) - 6\delta(f)^2$ increases with $\delta(f)$ only in the case that $\delta(f) \leq 1/4$. Furthermore, the lower-bound is useless in the case that $\delta(f) \geq 1/2$. Thus an alternative lower-bound is needed in case $\delta(f)$ approaches 1/2 (or is larger than it); see Exercise 9.21.

Exercise 9.21 (a better analysis of the BLR Linearity Test (cf. [40])) In continuation to Exercise 9.20, use the following guidelines in order to prove that $\varepsilon(f) \geq \min(1/6, \delta(f)/2)$. Specifically, focusing on the case that $\varepsilon(f) < 1/6$, show that f is $2\varepsilon(f)$-close to some homomorphism (and thus $\varepsilon(f) \geq \delta(f)/2$).

1. Define the vote of y regarding the value of f at x as $\phi_y(x) \stackrel{\rm def}{=} f(x+y) - f(y)$, and define $\phi(x)$ as the corresponding plurality vote (i.e., $\phi(x) \stackrel{\rm def}{=} \mathrm{argmax}_{v\in H}\{|\{y \in G : \phi_y(x) = v\}|\}$).

   Prove that, for every $x \in G$, it holds that $\mathsf{Pr}_y[\phi_y(x) = \phi(x)] \geq 1 - 2\varepsilon(f)$.

   Extra guideline: Fixing x, call a pair (y_1, y_2) good if $f(y_1) + f(y_2 - y_1) = f(y_2)$ and $f(x+y_1) + f(y_2 - y_1) = f(x+y_2)$. Prove that, for any x, a random pair (y_1, y_2) is good with probability at least $1 - 2\varepsilon(f)$. On the other hand, for a good (y_1, y_2), it holds that $\phi_{y_1}(x) = \phi_{y_2}(x)$. Show that the graph in which *edges* correspond to good pairs must have a connected component of size at least $(1 - 2\varepsilon(f)) \cdot |G|$. Note that $\phi_y(x)$ is identical for all vertices y in this connected component, which in turn contains a majority of all y's in G.

2. Prove that ϕ is a homomorphism; that is, prove that, for every $x, y \in G$, it holds that $\phi(x) + \phi(y) = \phi(x+y)$.

   Extra guideline: Prove that $\phi(x) + \phi(y) = \phi(x+y)$ holds by considering the somewhat fictitious expression $p_{x,y} \stackrel{\rm def}{=} \mathsf{Pr}_{r\in G}[\phi(x) + \phi(y) \neq \phi(x+y)]$, and showing that $p_{x,y} < 1$ (and hence $\phi(x) + \phi(y) \neq \phi(x+y)$ is false). Prove that $p_{x,y} < 1$, by showing that

$$p_{x,y} \leq \mathsf{Pr}_r \left[\begin{array}{l} \phi(x) \neq f(x+r) - f(r) \\ \vee\ \phi(y) \neq f(r) - f(r-y) \\ \vee\ \phi(x+y) \neq f(x+r) - f(r-y) \end{array} \right] \tag{9.10}$$

and using Item 1 (and some variable substitutions) for upper-bounding by $2\varepsilon(f) < 1/3$ the probability of each of the three events in Eq. (9.10).

3. Prove that f is $2\varepsilon(f)$-close to ϕ.

   Extra guideline: Denoting $B = \{x \in G : \mathsf{Pr}_{y\in G}[f(x) \neq \phi_y(x)] \geq 1/2\}$, prove that $\varepsilon(f) \geq (1/2)\cdot(|B|/|G|)$. Note that if $x \in G\setminus B$ then $f(x) = \phi(x)$.

We comment that better bounds on the behavior of $\varepsilon(f)$ as a function of $\delta(f)$ are known.

Exercise 9.22 (testing matrix identity) Let M be a non-zero m-by-n matrix over GF(p). Prove that $\mathsf{Pr}_{r,s}[r^\top Ms \neq 0] \geq (1-p^{-1})^2$, where r (resp., s) is a random m-ary (resp., n-ary) vector.

Guideline: Prove that if $v \neq 0^n$ then $\mathsf{Pr}_s[v^\top s = 0] = p^{-1}$, and that if M has rank ρ then $\mathsf{Pr}_r[r^\top M = 0^n] = p^{-\rho}$.

Exercise 9.23 (3SAT and CSP with two variables) Show that `3SAT` is reducible to $\mathtt{gapCSP}_\tau^{\{1,\ldots,7\}}$ for $\tau(m) = 1/m$, where `gapCSP` is as in Definition 9.18. Furthermore, show that the size of the resulting `gapCSP` instance is linear in the length of the input formula.

Guideline: Given an instance ψ of `3SAT`, consider the graph in which vertices correspond to clauses of ψ, edges correspond to pairs of clauses that share a variable, and the constraints represent the natural consistency condition regarding partial assignments that satisfy the clauses. See a similar construction in Exercise 9.18.

Exercise 9.24 (CSP with two Boolean variables) In contrast to Exercise 9.23, prove that for every positive function $\tau : \mathbb{N} \to (0,1]$ the problem $\mathtt{gapCSP}_\tau^{\{0,1\}}$ is solvable in polynomial-time.

Guideline: Reduce $\mathtt{gapCSP}_\tau^{\{0,1\}}$ to 2SAT.

Exercise 9.25 Show that, for any fixed finite Σ and constant $c > 0$, the problem $\mathtt{gapCSP}_c^\Sigma$ is in $\mathcal{PCP}(\mathtt{log}, O(1))$.

Guideline: Consider an oracle that, for some satisfying assignment for the CSP-instance (G, Φ), provides a trivial encoding of the assignment; that is, for a satisfying assignment $\alpha : V \to \Sigma$, the oracle responds to the query (v, i) with the i^{th} bit in the binary representation of $\alpha(v)$. Consider a verifier that uniformly selects an edge (u, v) of G and checks the constraint $\phi_{(u,v)}$ when applied to the values $\alpha(u)$ and $\alpha(v)$ obtained from the oracle. This verifier makes $\log_2|\Sigma|$ queries and reject each no-instance with probability at least c.

Exercise 9.26 For any constant Σ and $d \geq 14$, show that $\mathtt{gapCSP}^\Sigma$ can be reduced to itself such that the instance at the target of the reduction is a d-regular expander, and the fraction of violated constraints is preserved up to a constant factor. That is, the instance (G, Φ) is reduced to (G_1, Φ_1) such that G_1 is a d-regular expander graph and $\mathtt{vlt}(G_1, \Phi_1) = \Theta(\mathtt{vlt}(G, \Phi))$. Furthermore, make sure that $|G_1| = O(|G|)$ and that each vertex in G_1 has at least $d/2$ self-loops.

Guideline: First, replace each vertex of degree $d' > 3$ by a 3-regular expander of size d', and connect each of the original d' edges to a different vertex of this expander, thus obtaining a graph of maximum degree 4. Maintain the constraints associated with the original edges, and associate the equality constraint (i.e., $\phi(\sigma,\tau) = 1$ if and only if $\sigma = \tau$) to each new edge (residing in any of the added expanders). Next, augment the resulting N_1-vertex graph by the edges of a 3-regular expander of size N_1 (while associating with these edges the trivially satisfied constraint; i.e., $\phi(\sigma,\tau) = 1$ for all $\sigma, \tau \in \Sigma$). Finally, add at least $d/2$ self-loops to each vertex (using again trivially satisfied constraints), so to obtain a d-regular graph. Prove that this sequence of modifications may only decrease the fraction of violated constraints, and that the decrease is only by a constant factor. The latter assertion relies on the equality constraints associated with the small expanders used in the first step.

Exercise 9.27 (free-bit complexity zero) Note that only sets in co$\mathcal{RP}$ have PCPs of *query* complexity zero. Furthermore, Exercise 9.17 implies that only sets in $\mathcal{P}$ have PCP systems of logarithmic randomness and *query* complexity zero.

1. Show that only sets in $\mathcal{P}$ have PCP systems of logarithmic randomness and *free-bit* complexity zero.

   (Hint: Consider an application of the FGLSS-reduction to a set having a PCP of free-bit complexity zero.)

2. In contrast, show that Graph Non-Isomorphism has a PCP system of *free-bit* complexity zero (and linear randomness-complexity).

Exercise 9.28 (free-bit complexity one) In continuation to Exercise 9.27, prove that only sets in $\mathcal{P}$ have PCP systems of logarithmic randomness and free-bit complexity one.

Guideline: Consider an application of the FGLSS-reduction to a set having a PCP of free-bit complexity one and randomness-complexity r. Note that the question of whether the resulting graph has an independent set of size 2^r can be expressed as a 2CNF formula of size poly(2^r), and see Exercise 2.22.

Exercise 9.29 (Proving Theorem 9.23) Using the following guidelines, provide a proof of Theorem 9.23. Let $S \in \mathcal{NP}$ and consider the 3CNF formulae that are obtained by the standard reduction of S to `3SAT` (i.e., the one provided by the proofs of Theorems 2.21 and 2.22). Decouple the resulting 3CNF formulae into pairs of formulae (ψ_x, ϕ) such that ψ_x represents the "hard-wiring" of the input x and ϕ represents the computation itself. Referring to the mapping of 3CNF formulae to low-degree extensions presented in §9.3.2.2, show that the low-degree extension Φ that correspond to ϕ can be evaluated in polynomial-time (i.e., polynomial in the length of the input to Φ, which is $O(\log|\phi|)$). Conclude that the low-degree extension that corresponds to $\psi_x \wedge \phi$ can be evaluated in time $|x|^2$. Alternatively, note that it suffices to show that the assignment-oracle A (considered in §9.3.2.2) satisfies Φ and is consistent with x (and is a low-degree polynomial).

Guideline: Note that the circuit constructed in the proof of Theorem 2.21 is highly uniform. In particular, the relation between wires and gates in this circuit can be represented by constant-depth circuits of unbounded fan-in and polynomial-size (i.e., size that is polynomial in the length of the indices of wires and gates).

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 useful computational problems, it is natural to ask whether these problems can be relaxed such that the relaxation is both useful 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; specifically, we shall 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 hardness to the standard notion of worst-case hardness.

Organization. The first type of relaxation is treated in Section 10.1, where we consider approximations of search (or rather optimization) problems as well as approximate-decision problems (a.k.a property testing); see Section 10.1.1 and Section 10.1.2, respectively. The second type of relaxation, known as average/typical-case complexity, is treated in Section 10.2. The treatment of these two types is quite different. Section 10.1 provides a short and high-level introduction to various research areas, focusing on the main notions and illustrating them by reviewing some results (while providing no proofs). In contrast, Section 10.2 provides a basic treatment of a theory (of average/typical-case complexity), focusing on some basic results and providing a rather detailed exposition of the corresponding proofs.

10.1 Approximation

The notion of approximation is a very 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 will not provide a *general systematic classification* (because such a classification is not known).[1]

We shall distinguish between approximation problems that are of a "search type" and problems that 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 the distance between instances (of a decision problem).[2] We note that, sometimes 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 underlying both frameworks is that in each of them we extend the set of admissible solutions. In the case of search problems, we augment the set of optimal solutions by allowing 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 expressibility (or, actually, for greater flexibility), we allow the value of the solution to depend also on the instance itself.[3] 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

[1] In contrast, systematic classifications of restricted classes of approximation problems are known. For example, see [55] for a classification of (approximate versions of) Constraint Satisfaction Problems.

[2] In some sense, this distinction is analogous to the distinction between the two aforementioned uses of the word *approximation*.

[3] This convention is only a matter of convenience: without loss of generality, we can express the same optimization problem using a value function that only depends on the solution by augmenting each solution with the corresponding instance (i.e., a solution y to an instance x can be encoded as a pair (x, y), and the resulting set of valid solutions for x will consist of pairs of the form $(x, \cdot)$). Hence, the foregoing convention merely allows avoiding this cumbersome encoding of solutions.

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. 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).[4] 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 $g : \{0,1\}^* \to \{r \in \mathbb{R} : r \geq 1\}$.*

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 $G = (V,E)$*

[4] However, in this case (and in contrast to Footnote 3), the value function f must depend both on the instance and on the solution (i.e., $f(x,y)$ may no be oblivious of x).

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 search problem (of `maximum Independent Set`). This phenomenon stands in contrast to the case of optimization, where an optimal solution for one search problem (e.g., `minimum Vertex Cover`) yields an optimal solution for the complementary search 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 satisfy some natural constraints (and pretty good approximations are feasible).

Theorem 10.3 (approximations to special cases of `TSP`): *Polynomial-time algorithms exist 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 exemplified 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* (abbreviated 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 a general graph $G=(V,E)$ is as hard as finding a maximum clique in a general graph. This follows from the fact that the approximation problem is NP-hard (cf. Theorem 10.5).

The statement of such 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 inapproximability result 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 at least $g_1(x)$ 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. □

Additional examples. Let us consider $\mathtt{gapVC}_{s,L}$, 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 $\mathtt{gapVC}_{s,2s}$ 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 $s > 0$ and $L < 1$ such that $L > \frac{4}{3} \cdot s$* (e.g., $s = 0.62$ and $L = 0.84$), *the problem $\mathtt{gapVC}_{s,L}$ 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, $\mathtt{gapSAT}^3_\varepsilon$ 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 $\mathtt{gapSAT}^3_v$ is NP-hard.*

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

Sharp thresholds. The aforementioned opposite results (regarding $\texttt{gapSAT}_v^3$) 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 (for every $L \geq 0.5$) 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 thresholds). 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 computationally 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 relative distance*: Fixing a distance parameter δ, we consider the problem of distinguishing inputs in S from inputs in $\Gamma_\delta(S)$, where
$$\Gamma_\delta(S) \stackrel{\text{def}}{=} \{x : \forall z \in S \cap \{0,1\}^{|x|} \;\; \Delta(x,z) > \delta \cdot |x|\} \tag{10.1}$$
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 (relative) 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, in polylogarithmic time), when given direct access to the input. A few examples will be presented in §10.1.2.2 (but, as indicated in §10.1.2.2, this is not a generic phenomenon). Before turning to these examples, 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 with 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 §1.2.3.6)*. In addition, the algorithm is given the length of the input and possibly other parameters on a* 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. In the current model, polylogarithmic time means time that is polylogarithmic in the length of the main input, which means time that is polynomial in the length of the binary representation of the length of the main input. Thus, polylogarithmic time yields a robust notion of extremely efficient computations. As we shall see, such computations suffice for solving various (property testing) problems.

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 graph was represented by its adjacency matrix or by its incidence-list. 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, $\Gamma_{0.1}(S)$. 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 explicitly 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*[5] *with at least $\rho \cdot N^2$ edges.*
- *For every fixed $\rho > 0$, the set of N-vertex graphs having a bisection*[5] *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 (asserted in Theorem 10.12) use a constant number of queries, where this constant is polynomial in the constant $1/\delta$. In contrast, exact decision procedures 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 (asserted in Theorem 10.12), we mention that the graph property for which this bound is proved is not a natural one. As in §10.1.2.1, 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

[5] A cut in a graph $G = ([N], E)$ is a partition (S_1, S_2) of the set of vertices (i.e., $S_1 \cup S_2 = [N]$ and $S_1 \cap S_2 = \emptyset$), and the edges of the cut are the edges with exactly one endpoint in S_1. A bisection is a cut of the graph to two parts of equal cardinality.

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.[6] 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). For example, property testing for the set of *triangle-free* graphs is possible by using a number of queries that depends only on δ, but it is known that this number must grow faster than any polynomial in $1/\delta$.

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 subgraph of a given size.

Indeed, Theorem 10.12 is meaningful only for dense graphs. This holds, in general, for any graph property in the adjacency matrix representation.[7] 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).

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 time randomized algorithm that solves the property testing problem for the set of bipartite graphs of degree*

[6] 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.

[7] In this model, as shown next, property testing of non-dense graphs is trivial. Specifically, fixing the distance parameter δ, we call a N-vertex graph non-dense if it has less than $(\delta/2) \cdot \binom{N}{2}$ edges. The point is that, for non-dense graphs, the property testing problem for any set S is trivial, because we may just accept any non-dense (N-vertex) graph if and only if S contains some non-dense (N-vertex) graph. Clearly, the decision is correct in the case that S does not contain non-dense graphs. However, the decision is admissible also in the case that S does contain some non-dense graph, because in this case every non-dense graph is "δ-close" to S (i.e., it is not in $\Gamma_\delta(S)$).

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 of degree at most d requires $\Omega(N)$ queries.*

The running time of the algorithms (asserted in Theorem 10.13) 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 type of computational problems 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 ("full" or "explicit") description of an m-variate function over $\mathrm{GF}(q)$, 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 $q = \mathrm{poly}(d)$ 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 rather typical) instances, and not as the average performance on random instances. This choice is justified in §10.2.1.1. Thus, it may be more justified to refer to the following theory by the name typical-case complexity. Still, the name average-case was 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; that is, 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. Most importantly, 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 in a theory of average-case complexity 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 relevant 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. The study of average-case complexity is aimed at exploring the possible benefit of such a relaxation, distinguishing cases in which a benefit exists from cases in which it does not exist. A key aspect in such a study is a good modeling of the type of distributions (of instances) that are encountered in natural algorithmic applications.

A preliminary question that arises is *whether every natural computational problem be solve efficiently when restricting attention to typical instances?* The conjecture that underlies this section is that, for a well-motivated choice of definitions, the answer is negative; that is, 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 are 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).[8]

The foregoing 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. In particular, this (average-case) theory identifies distributional problems that are "typically hard" provided that distributional problems that are "typically hard" exist at all. If one believes the foregoing conjecture then, for such complete (distributional) problems, one should not seek algorithms that solve these problems efficiently on typical instances.

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 distributional problems that are "NP-complete" in the corresponding average-case complexity sense. Furthermore, we show how to obtain such a distributional version for any natural NP-complete decision problem. In §10.2.1.3 we extend the treatment to randomized algorithms. Additional ramifications are presented in Section 10.2.2.

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

[8]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*.

well as the identification of complete problems 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 at first thought. In addition to the generic conceptual 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 hardness to worst-case hardness) 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.[9] For simplicity, here a probability ensemble $\{X_n\}_{n\in\mathbb{N}}$ is a 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 $\{0,1\}^n$.

[9] We mention that even this choice is not evident. Specifically, Levin [153] (see discussion in [88]) 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 [88]), but currently we feel that the representation-dependent effects discussed in [88] are legitimate. Furthermore, the alternative formulation of [153, 88] comes across as unnatural and tends to confuse some readers.

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.[10]

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

We stress 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

[10] An alternative choice, taken by Levin [153] (see discussion in [88]), 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 $t(x) \leq p(\ell(x))$ for every 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., $\mathsf{E}[t_A(X_n)] \leq \text{poly}(n)$).

is 3-colorable. Intuitively, this problem is "typically trivial" (with respect to the uniform distribution),[11] because the algorithm may always say no 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 discussions, 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.*[12] *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 all *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 hardness to worst-case hardness (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.[13] 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 $x \in \{0,1\}^*$, outputs* $\Pr[X_{|x|} \le x]$, *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.*

---

[11] In contrast, testing whether a given graph is 3-colorable seems "typically hard" for other distributions (see either Theorem 10.19 or Exercise 10.27). Needless to say, in the latter distributions both yes-instances and no-instances appear with noticeable probability.

[12] 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 $x \in S$.

[13] Confining attention to the uniform distribution seems misguided by the naive belief according to which this distribution is the *only* one relevant to applications. In contrast, we believe that, for most natural applications, the uniform distribution over instances is not relevant at all.

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).

Teaching note: The following two paragraphs attempt to address some doubts regarding Definition 10.15. One may postpone such discussions to a later stage.

We admit that the identification of simple distributions as the class of interesting distribution is significantly more questionable than any other identification advocated in this book. Nevertheless, we believe that we were fully justified in rejecting both the aforementioned extremes (i.e., of either allowing all distributions or allowing only the uniform distribution). Yet, the reader may wonder whether or not we have struck the right balance between "generality" and "simplicity" (in the intuitive sense). One specific concern is that we might have restricted the class of distributions too much. We briefly address this concern next.

A more intuitive and very robust class of distributions, which seems to contain all distributions that may occur in applications, is the class of polynomial-time sampleable probability ensembles (treated in §10.2.2.2). Fortunately, the combination of the results presented in §10.2.1.2 and §10.2.2.2 seems to retrospectively endorse the choice underlying Definition 10.15. Specifically, 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 more appealing when the problem belongs to a smaller class that corresponds to a restricted definition of admissible distributions. Now, 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 to be 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 leads to 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. In particular, we should not expect to see a proof that some specific problem in dist$\mathcal{NP}$ is not in tpc$\mathcal{P}$. 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.

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 15.

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.*[14]

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:[15] *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 $p'(|y|) \geq |x|$.

[14] 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.

[15] 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 $\mathsf{Pr}[f(X_n) = y]$. That is, Eq. (10.2) simplifies to $\mathsf{Pr}[f(X_n) = y] \leq p(|y|) \cdot \mathsf{Pr}[Y_{|y|} = y]$. Indeed, this condition holds vacuously for any y that is not in the image of f.

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 $\Pr[Y_{|y|} = y]$.

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 $Q(X)$ equals Y, but rather allows some slackness that, in turn, is bounded so to guarantee preservation of typical feasibility (see Exercise 10.15).[16]

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 that 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 somewhat "bizarre" instances of the second problem. Thus, the current subsection may be viewed as a study of reductions that do not commit this sin.[17]

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 reductions are via many-to-one mappings.*

Proof: We start by introducing such a (distributional) problem, which is a natural distributional version of the decision problem $S_{\mathbf{u}}$ (used in the proof of

[16]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 14) suffices for the preservation of typical feasibility.

[17]The latter assertion is somewhat controversial. While it seems totally justified with respect to the proof of Theorem 10.17, opinions regarding the proof of Theorem 10.19 may differ.

Theorem 2.19). Recall that $S_{\mathtt{u}}$ contains the instance $\langle M, x, 1^t\rangle$ if there exists $y \in \cup_{i \le t}\{0,1\}^i$ such that machine 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

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

where $n \stackrel{\text{def}}{=} |\langle M, x, 1^t\rangle| \stackrel{\text{def}}{=} |M| + |x| + t$. Note that, under a suitable natural encoding, the ensemble U' is indeed simple.[18]

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 2.19), 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., $(S_{\mathtt{u}}, U')$). In contrast, considering the distributions induced by 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 (of S to $S_{\mathtt{u}}$), when applied to an arbitrary $(S, X) \in \text{dist}\mathcal{NP}$. 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,

---

[18] 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 $[k+\ell+t]$.

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 $(S, X) \in \text{dist}\mathcal{NP}$ is reducible to (S_{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 $\mu(x) \stackrel{\text{def}}{=} \Pr[X_{|x|} \le x]$, and note that $\mu : \{0,1\}^* \to [0,1]$ is polynomial-time computable (because X satisfies Definition 10.15).

Coding Lemma:[19] 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., $\mu(x') \le \mu(x'')$ 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 $|C_\mu(x)| \le 1 + \min\{|x|, \log_2(1/\mu'(x))\}$, where $\mu'(x) \stackrel{\text{def}}{=} \mu(x) - \mu(x-1)$ if $x \notin \{0\}^*$ and $\mu'(0^n) \stackrel{\text{def}}{=} \mu(0^n)$ 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 $\mu'(x) \le 2^{-|x|}$ 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 $|z| \le \log_2(1/\mu'(x))$ 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., $\mu(x-1) < 0.z1 \le \mu(x)$). 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

[19] 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.

$\mu'(x) > 2^{-|x|}$ 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

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

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 $\mu(x-1) < 0.z1 \leq \mu(x)$. □

In order 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 $C'_\mu(x) \stackrel{\text{def}}{=} (|x|, C_\mu(x))$, 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 (over $\{0,1\}^*$).

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 $y \in \{0,1\}^{\text{poly}(|x|)}$ 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 $C'_\mu(x)$).[20]

The reduction. We maps an instance x (of S) to the triple $(M_{S,\mu}, C'_\mu(x), 1^{p(|x|)})$, where $p(n) \stackrel{\text{def}}{=} p_S(n) + p_C(n)$ 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.

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

Analyzing the reduction. Our goal is proving that *the foregoing mapping constitutes a reduction of* (S, X) *to* $(S_{\mathbf{u}}, U')$. 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 $C'_\mu(x) \in S'$, which in turn holds if and only if there exists a string y such that $M_{S,\mu}$ accepts $(C'_\mu(x), \langle x, y\rangle)$ in at most $p(|x|)$ 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

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

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

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

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

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

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 highlights the fact that the reduction used in the proof of Theorem 2.19 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 2.19 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).[21]

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 version of $S_{\mathbf{u}}$. 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 known 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). Actually, as demonstrated next, the problem is not the existence of a structure in the reduced instances but rather the complexity of this structure. In particular, if the aforementioned reduction is "monotone" and "length regular" then the distribution of the reduced instances is simple enough (i.e., is simple in the sense of Definition 10.15):

Proposition 10.18 (sufficient condition for dist$\mathcal{NP}$-completeness): *Suppose that f is a Karp-reduction of the set S to the set T such that, for every $x', x'' \in \{0,1\}^*$, the following two conditions hold:*

1. (f is monotone): *If $x' < x''$ then $f(x') < f(x'')$, where the inequalities refer to the standard lexicographic order of strings.*[22]
2. (f is length-regular): *$|x'| = |x''|$ if and only if $|f(x')| = |f(x'')|$.*

Then if there exists an ensemble X such that (S, X) is dist$\mathcal{NP}$*-complete then there exists an ensemble Y such that (T, Y) is* dist$\mathcal{NP}$*-complete.*

Proof Sketch: Note that the monotonicity of f implies that f is one-to-one and that for every x it holds that $f(x) \geq x$. Furthermore, as shown next, f is polynomial-time invertible. Intuitively, the fact that f is both monotone and

[21] 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 2.31.

[22] In particular, if $|z'| < |z''|$ then $z' < z''$. Recall that for $|z'| = |z''|$ it holds that $z' < z''$ if and only if there exists $w, u', u'' \in \{0,1\}^*$ such that $z' = w0u'$ and $z'' = w1u''$.

polynomial-time computable implies that a preimage can be found by a binary search. Specifically, given $y = f(x)$, we search for x by iteratively halving the interval of potential solutions, which is initialized to $[0, y]$ (since $x \leq f(x)$). Note that if this search is invoked on a string y that is not in the image of f, then it terminates while detecting this fact.

Relying on the fact that f is one-to-one (and length-regular), we define the probability ensemble $Y = \{Y_n\}_n$ such that for every x it holds that $\Pr[Y_{|f(x)|} = f(x)] = \Pr[X_{|x|} = x]$. Specifically, letting $\ell(m) = |f(1^m)|$ and noting that ℓ is one-to-one and monotonically non-decreasing, we define

$$\Pr[Y_{|y|} = y] = \begin{cases} \Pr[X_{|x|} = x] & \text{if } x = f^{-1}(y) \\ 0 & \text{if } \exists m \text{ s.t. } y \in \{0,1\}^{\ell(m)} \setminus \{f(x) : x \in \{0,1\}^m\} \\ 2^{-|y|} & \text{otherwise (i.e., if } |y| \notin \{\ell(m) : m \in \mathbb{N}\})^{23}. \end{cases}$$

Clearly, (S, X) is reducible to (T, Y) (via the Karp-reduction f, which, due to our construction of Y, also satisfies the domination condition). Thus, using the hypothesis that dist$\mathcal{NP}$ is reducible to (S, X) and the transitivity of reductions (see Exercise 10.17), it follows that every problem in dist$\mathcal{NP}$ is reducible to (T, Y). The key observation, to be established next, is that Y is a simple probability ensemble, and it follows that (T, Y) is in dist$\mathcal{NP}$.

Loosely speaking, the simplicity of Y follows by combining the simplicity of X and the properties of f (i.e., the fact that f is monotone, length-regular, and polynomial-time invertible). The monotonicity and length-regularity of f implies that $\Pr[Y_{|f(x)|} \leq f(x)] = \Pr[X_{|x|} \leq x]$. More generally, for any $y \in \{0,1\}^{\ell(m)}$, it holds that $\Pr[Y_{\ell(m)} \leq y] = \Pr[X_m \leq x]$, where x is the lexicographicly largest string such that $f(x) \leq y$ (and, indeed, if $|x| < m$ then $\Pr[Y_{\ell(m)} \leq y] = \Pr[X_m \leq x] = 0$).[24] Note that this x can be found in polynomial-time by the inverting algorithm sketched in the first paragraph of the proof. Thus, we may compute $\Pr[Y_{|y|} \leq y]$ by finding the adequate x and computing $\Pr[X_{|x|} \leq x]$. Using the hypothesis that X is simple, it follows that Y is simple (and the proposition follows). □

On the existence of adequate Karp-reductions. Proposition 10.18 implies that a sufficient condition for the dist$\mathcal{NP}$-completeness of a distributional version of a (NP-complete) set T is the existence of an adequate Karp-reduction from the set $S_{\mathbf{u}}$ to the set T; that is, this Karp-reduction should be monotone and length-regular. While the length-regularity condition seems easy to impose (by using adequate padding), the monotonicity condition seems more problematic. Fortunately, it turns out that the monotonicity condition can also be imposed by using adequate padding (or rather an adequate "marking" – see Exercises 2.30 and 10.21). We highlight the fact that the existence of an adequate padding (or "marking") is a property of the set T itself. In Exercise 10.21 we review a method for modifying any Karp-reduction to a "monotonically markable" set T into a Karp-reduction (to

[23] Having Y_n be uniform in this case is a rather arbitrary choice, which is merely aimed at guaranteeing a "simple" distribution on n-bit strings (also in this case).

[24] We also note that the case in which $|y|$ is not in the image of ℓ can be easily detected and taken care off accordingly.

T) that is monotone and length-regular. In Exercise 10.23 we provide evidence to the thesis that all natural NP-complete sets are monotonically markable. Combining all these facts, we conclude that *any natural NP-complete decision problem can be coupled with a simple probability ensemble such that the resulting distributional problem is* dist$\mathcal{NP}$*-complete*. As a concrete illustration of this thesis, we state the corresponding (formal) result for the twenty-one NP-complete problems treated in Karp's paper on NP-completeness [136].

Theorem 10.19 (a modest version of a general thesis): *For each of the twenty-one NP-complete problems treated in [136] there exists a simple probability ensemble such that the combined distributional problem is* dist$\mathcal{NP}$*-complete.*

The said list of problems includes `SAT`, `Clique`, and `3-Colorability`.

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 10.20 (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 10.24).

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., $\Pr[U'_n = \langle M, x, 1^t\rangle] = 2^{-(|M|+|x|)}/\binom{n}{2}$, 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 $\Pr[U''_n = \langle M, x, z\rangle] = 2^{-(|M|+|x|+|z|)}/\binom{n}{2}$ 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 10.21 *There exists $S \in \mathcal{NP}$ such that every $(S', X') \in$ 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 $\text{bin}_\ell(|x|)\cdot\text{bin}_\ell(|M|)\cdot Mxz$, where $\ell = \lceil \log_2(|M| + |x| + |z|)\rceil + 1$. Noting that this reduction satisfies all conditions of Definition 10.16, the proposition follows. □

10.2.2 Ramifications

In our opinion, the most problematic aspect of the theory described in Section 10.2.1 is the choice to focus on simple probability ensembles, which in turn restricts "distributional versions of NP" to the class $\text{dist}\mathcal{NP}$ (Definition 10.15). As indicated §10.2.1.1, this restriction raises two opposite concerns (i.e., that $\text{dist}\mathcal{NP}$ is either too wide or too narrow).[25] Here we address the concern that the class of simple probability ensembles is too restricted, and consequently that the conjecture $\text{dist}\mathcal{NP} \not\subseteq \text{tpc}\mathcal{BPP}$ is too strong (which would mean that $\text{dist}\mathcal{NP}$-completeness is a weak evidence for typical-case hardness). An appealing extension of the class of simple probability ensembles is presented in §10.2.2.2, yielding an corresponding extension of $\text{dist}\mathcal{NP}$, and it is shown that if this extension of $\text{dist}\mathcal{NP}$ is not contained in $\text{tpc}\mathcal{BPP}$ then $\text{dist}\mathcal{NP}$ itself is not contained in $\text{tpc}\mathcal{BPP}$. Consequently, $\text{dist}\mathcal{NP}$-complete problems enjoy the benefit of both being in the more restricted class (i.e., $\text{dist}\mathcal{NP}$) 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. In addition, the technical development of §10.2.2.2 relies on §10.2.2.1.

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

[25]On one hand, if the definition of $\text{dist}\mathcal{NP}$ were too liberal then membership in $\text{dist}\mathcal{NP}$ would mean less than one may desire. On the other hand, if $\text{dist}\mathcal{NP}$ were too restricted then the conjecture that $\text{dist}\mathcal{NP}$ contains hard problems would have been very questionable.

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 10.22 (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 10.20). 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 10.23 (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 2.16). 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 6.28); 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^{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 $(R, X) \in \text{dist}\mathcal{PC}$ (having unique solutions) to $(S''_R, X'') \in \text{dist}\mathcal{NP}$, *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 6.29, this difficulty can be overcome. We first note that the foregoing mapping of instances of the distributional problem $(R, X) \in \text{dist}\mathcal{PC}$ to instances of $(S''_R, X'') \in \text{dist}\mathcal{NP}$ 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 6.29 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 $\ell = \text{poly}(|x|)$ 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 $R'(x, m, h) \stackrel{\text{def}}{=} \{y \in R(x) : h(y) = 0^m\}$, this yields a randomized reduction of the search problem of R to the search problem of R' such that with noticeable probability[26] the reduction maps instances that have solutions to instances having a unique solution. Furthermore, this reduction can be used to reduce any $(R, X) \in \text{dist}\mathcal{PC}$ to $(R', X') \in \text{dist}\mathcal{PC}$, *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 10.24). □

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.

[26] 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.

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 $\mu(x) \stackrel{\rm def}{=} \Pr[X_n \le x]$, 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 "simple" in some intuitive sense, and that they represent a reasonable (alas disputable) model of distributions that may occur in practice. Still, in light of the fear that this model is too restrictive (and consequently that dist$\mathcal{NP}$-hardness is weak evidence for typical-case hardness), we seek a maximalistic model of distributions that may occur in practice. Such a model is provided by the notion of polynomial-time sampleable ensembles (underlying Definition 10.24). Our maximality thesis is based on the belief that the real world should be modeled as a *feasible* randomized process (rather than as an arbitrary process). This belief implies that all objects encountered in the world may be viewed as samples generated by a feasible randomized process.

Definition 10.24 (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 10.25), and thus dist$\mathcal{NP} \subseteq$ samp$\mathcal{NP}$. On the other hand, there exist sampleable probability ensembles that do not seem simple (see Exercise 10.26).

Extending the scope of distributional problems (from dist$\mathcal{NP}$ to samp$\mathcal{NP}$) facilitates the presentation of complete distributional problems. We first note that it is easy to prove that every natural NP-complete problem has a distributional version in samp$\mathcal{NP}$ that is dist$\mathcal{NP}$-hard (see Exercise 10.27). Furthermore, it is possible to prove that all natural NP-complete problem have distributional versions that are samp$\mathcal{NP}$-complete. (In both cases, "natural" means that the corresponding Karp-reductions do not shrink the input, which is a weaker condition than the one in Proposition 10.18.)

Theorem 10.25 (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 10.25 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 10.28.

Theorem 10.25 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

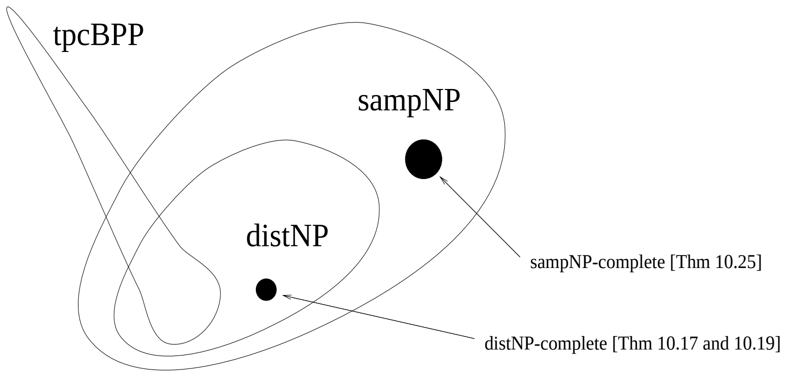


Figure 10.1: Two types of average-case completeness

Figure 10.1). This is essentially 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 10.26 (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 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 10.26 is a more liberal notion of a reduction among distributional problems.

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$\mathcal{BPP}$). Relying on Theorem 10.23 and Exercise 10.29, it suffices to 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$\mathcal{BPPF}$. This will be shown by showing 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.[27] We start by formulating this notion, when referring to distributional *search* problems.

[27] 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

> **Teaching note:** The following proof is quite involved and is better left for advanced reading. Its main idea is related to 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 additional motivation for reading the following proof.

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 with respect to a family of sets $\{\Omega_x \subseteq \{0,1\}^{m(|x|)} : x \in \{0,1\}^*\}$, where $m(|x|) = \text{poly}(|x|)$ denotes an upper-bound on the number of the internal coin tosses of M on input x:

Density (of Ω_x): There exists a noticeable function $\rho : \mathbb{N} \to [0,1]$ (i.e., $\rho(n) > 1/\text{poly}(n)$) such that, for every $x \in \{0,1\}^*$, it holds that $|\Omega_x| \geq \rho(|x|) \cdot 2^{m(|x|)}$.

Validity (with respect to Ω_x): For every $r \in \Omega_x$ the reduction yields a correct answer; that is, $M^T(x,r) \in R(x)$ if $R(x) \neq \emptyset$ and $M^T(x,r) = \perp$ otherwise, where $M^T(x,r)$ denotes the execution of M on input x, internal coins r, and oracle access to T.

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

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

where $Q'(x)$ is a random variable, defined over the set Ω_x, representing the set of queries made by M on input x, coins in Ω_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 $R \in \mathcal{PC}$.

repeated invocations of the reduction, such a process will *not* redeem the violation of the standard domination condition.

Our goal is presenting, for every $(R,X) \in \text{samp}\mathcal{PC}$, a relaxed reduction of (R,X) to a related problem $(R',X') \in \text{dist}\mathcal{PC}$. (We use the standard notation $X = \{X_n\}_{n\in\mathbb{N}}$ 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 $\text{dist}\mathcal{PC}$ (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 $R'(h, 1^{n-\ell(n)}0v) = R(x)$ such that 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 desired solution $y \in R(x)$ if $S(1^{|x|},r) = x$, but otherwise "all bets are off" (since 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 6.29.) 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 (on input x, 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 $\ell(n) = \log_2(1/\Pr[X_n = x])$ 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, we 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

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

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 efficiently 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., $\Pr[\ell=\ell_x]=1/(q(|x|)+1)=1/\mathrm{poly}(|x|))$. 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.[28] This mapping will be used to reduce (R,X) to (R',X'), where R' and $X'=\{X'_{n'}\}_{n'\in\mathbb{N}}$ 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 foregoing 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 $h'(r)=v'$. Furthermore, with constant probability over the choice of $h'\in H_{q(n)}^{q(n)-\ell_x}$ and $v'\in\{0,1\}^{q(n)-\ell_x}$, it also holds that there are at most $O(2^{\ell_x})$ strings r such that $h'(r)=v'$. Fixing such h' and v', we let $S_{h',v'}=\{S(1^n,r) : h'(r)=v'\}$ and we note that, with constant probability over the choice of $h\in H_n^{\ell_x}$, it holds that x is the only string in $S_{h',v'}$ that is mapped to $h(x)$ under h. Thus, 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 $\{r : h(S(1^n,r))=h(x)\wedge h'(r)=v'\}\neq\emptyset$. The theorem follows. □

Reflection: Theorem 10.26 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.

[28] 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 $n'=\mathrm{poly}(n)$). For example, we may encode $\langle 1^n,1^\ell,h,h(x),h',v'\rangle$ as $1^n01^\ell01^{q(n)-\ell}0\langle h\rangle\langle h(x)\rangle\langle h'\rangle\langle v'\rangle$, where each $\langle w\rangle$ denotes an encoding of w by a string of length $(n'-(n+q(n)+3))/4$.

Furthermore, by Proposition 10.21, 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 $R_f = \{(f(r), r) : r \in \{0,1\}^*\}$.[29] 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.

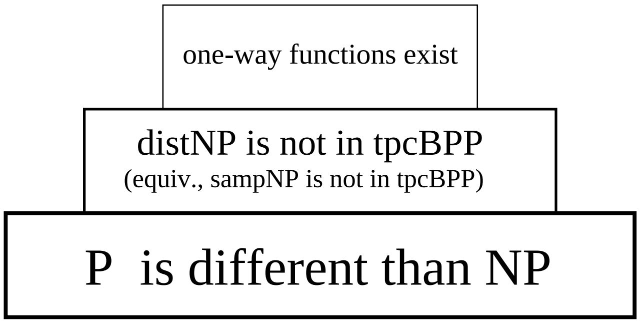


Figure 10.2: Worst-case vs average-case assumptions

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 least-standard framework we used (and it suffices for casting gap problems and

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

property testing). In contrast, the study of average-case complexity requires the introduction of a new conceptual framework and addressing 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 versions). 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 NP-complete problem has a distributional version that is (typical-case) hard, where this version refers to a sampleable ensemble (and, in fact, even to a simple ensemble). Furthermore, in this case, some problems in NP have hard distributional versions that refer to the uniform distribution.

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 [178]), 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 [72]. Indeed the relatively "tight" inapproximation results for max-Clique, max-SAT, and the maximization of linear equations, due to Håstad [115, 116], build on previous work regarding PCP and their connection to approximation (cf., e.g., [73, 15, 14, 28, 184]). Specifically, Theorem 10.5 is due to [115][30], while Theorems 10.8 and 10.9 are due to [116]. The best known inapproximation result for minimum Vertex Cover (see Theorem 10.7) is due to [68], but we doubt it is tight (see, e.g., [142]). Reductions among approximation problems were defined and presented in [178]; see Exercise 10.7, which presents a major technique introduced in [178]. 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 [121]. A compendium of NP optimization problems is available at [63].

Recall that a different type of approximation problems, which are naturally associated with search problems, refer to approximately counting the number of solutions. These approximation problems were treated in Section 6.2.2 in a rather *ad hoc* manner. We note that a more systematic treatment of approximate counting problems can be obtained by using the definitional framework of Section 10.1.1 (e.g., the notions of gap problems, polynomial-time approximation schemes, etc).

[30] See also [242].

Property testing. The study of property testing was initiated by Rubinfeld and Sudan [194] and re-initiated by Goldreich, Goldwasser, and Ron [96]. While the focus of [194] was on algebraic properties such as low-degree polynomials, the focus of [96] was on graph properties (and Theorem 10.12 is taken from [96]). The model of bounded-degree graphs was introduced in [102] and Theorem 10.13 combines results from [102, 103, 41]. For surveys of the area, the interested reader is referred to [76, 193].

Average-case complexity

The theory of average-case complexity was initiated by Levin [153], who in particular proved Theorem 10.17. In light of the laconic nature of the original text [153], we refer the interested reader to a survey [88], 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 [88] 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 [43].

Turning to §10.2.1.2, we note that while the existence of dist$\mathcal{NP}$-complete problems (cf. Theorem 10.17) was established in Levin's original paper [153], the existence of dist$\mathcal{NP}$-complete versions of all natural NP-complete decision problems (cf. Theorem 10.19) was established more than two decades later in [157].

Section 10.2.2 is based on [29, 126]. Specifically, Theorem 10.23 (or rather the reduction of search to decision) is due to [29] and so is the introduction of the class samp$\mathcal{NP}$. A version of Theorem 10.26 was proven in [126], and our proof follows their ideas, which in turn are closely related to the ideas underlying the proof of Theorem 8.11 (proved in [117]).

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 distributional problems do not seem very natural (i.e., they either refer to somewhat generic decision problems such as $S_{\mathtt{u}}$ or to somewhat contrived probability ensembles (cf. Theorem 10.19)). The presentation of dist$\mathcal{NP}$-complete problems that combine a more natural decision problem (like `SAT` or `Clique`) with a more natural probability ensemble is an open problem.

Exercises

Exercise 10.1 (general TSP) For any adequate 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. Specifically, show that the result holds with $g(I) = \exp(|I|^{0.99})$ and for instances in which all distances are positive integers.

Guideline: Use a reduction from Hamiltonian cycle problem. 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 9.18), while noting that x is reduced to a graph of size $\text{poly}(|x|)$ 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 9.18 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 $\texttt{gapSAT}^3_\varepsilon$ 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 2.24, 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 2.25), 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 2.25, 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 8.5.2), 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 technique (of Papadimitriou and Yannakakis [178]) that is useful for designing reductions among approximation problems. Recalling that $\mathtt{gapSAT}^3_{0.1}$ is NP-hard, our goal is proving NP-hard of the following gap problem, denoted $\mathtt{gapSAT}^{3,c}_{\varepsilon}$, which is a special case of $\mathtt{gapSAT}^3_{\varepsilon}$. 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 $\mathtt{3SAT}$ to the corresponding special case (see proof of Proposition 2.23) does not preserve an approximation gap.[31] 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 an m-vertex expander graph. Prove that, for some constants c and $\varepsilon > 0$, the corresponding mapping reduces $\mathtt{gapSAT}^3_{0.1}$ to $\mathtt{gapSAT}^{3,c}_{\varepsilon}$.

Guideline: Using d-regular expanders in the foregoing reduction, we map general 3CNF formulae to 3CNF formulae 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.

[31] 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 10.8 (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., $\lfloor n/2 \rfloor + 1$) zeros. 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 (in the sense of Definition 10.11) can be done in polylogarithmic time by probing the input at a constant number of randomly selected locations.

Exercise 10.10 (on the triviality of some testing problems) Show that the following sets are trivially testable in the adjacency matrix representation (i.e., for every $\delta > 0$ and any such set S, there exists a trivial algorithm that distinguishes S from $\Gamma_\delta(S)$).

1. The set of connected graphs.
2. The set of Hamiltonian graphs.
3. The set of Eulerian graphs.

Indeed, show that in each case $\Gamma_\delta(S) = \emptyset$.

Guideline (for Item 3): 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 $\mathrm{tpc}\mathcal{P}$**)** Prove that $(S, X) \in \mathrm{tpc}\mathcal{P}$ 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 ($\mathrm{tpc}\mathcal{P}$ **versus** $\mathcal{P}$ **– Part 1)** Prove that $\mathrm{tpc}\mathcal{P}$ 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 ($\mathrm{tpc}\mathcal{P}$ **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 $S' \in \mathrm{DTIME}(t) \setminus \mathcal{P}$.

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 $\Pr[X_{|x|} \leq x]$. (Thus, the binary representation of $\Pr[X_{|x|} \leq x]$ has length at most $p(|x|)$.) The desired function $f : \{0,1\}^{p(n)} \to \{0,1\}^n$ is obtained by defining $f(r) = x$ if the number (represented by) $0.r$ resides in the interval $[\Pr[X_n < x], \Pr[X_n \leq x])$. 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 $\Pr[Q(X_n) \cap B' \neq \emptyset]$ is a negligible function (in n), using both $\Pr[y \in Q(X_n)] \leq p(|y|) \cdot \Pr[X'_{|y|} = y]$ and $|x| \leq p'(|y|)$ for every $y \in Q(x)$. Specifically, use the latter condition for inferring that $\sum_{y \in B'} \Pr[y \in Q(X_n)]$ equals $\sum_{y \in \{y' \in B' : p'(|y'|) \geq n\}} \Pr[y \in Q(X_n)]$, which is upper-bounded by $\sum_{m : p'(m) \geq n} p(m) \cdot \Pr[X'_m \in B']$ (which in turn is negligible in terms of 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 2.19, 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 $\mu'(x) \stackrel{\rm def}{=} \mu(x) - \mu(x-1)$.

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 dist$\mathcal{NP}$ there exists a simple probability ensemble Y such that the reduction used in the proof of Theorem 2.19 suffices for reducing (S,X) to $(S_{\tt u}, Y)$.

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 $\pi_x \stackrel{\rm def}{=} \Pr[X_{|x|} = x]$. 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 2.19.

Exercise 10.21 (monotone markability and monotone reductions) In continuation to Exercise 2.30, we say that a set T is monotonically markable if there exists a polynomial-time (marking) algorithm M such that

1. For every $z, \alpha \in \{0,1\}^*$, it holds that $M(z,\alpha) \in T$ if and only if $z \in T$.

2. Monotonicity: for every $|z'| = |z''|$ and $\alpha' < \alpha''$, it holds that $M(z',\alpha') < M(z'',\alpha'')$, where the inequalities refer to the standard lexicographic order of strings.

3. Auxiliary length requirements:

   (a) If $|z'| = |z''|$ and $|\alpha'| = |\alpha''|$, then $|M(z',\alpha')| = |M(z'',\alpha'')|$.

   (b) If $|z'| \leq |z''|$ and $|\alpha'| < |\alpha''|$, then $|M(z',\alpha')| < |M(z'',\alpha'')|$.

   (c) There exists a 1-1 polynomial $p : \mathbb{N} \to \mathbb{N}$ such that for every ℓ and every $z \in \cup_{i=1}^{\ell}\{0,1\}^i$ there exists $t \in [p(\ell)]$ such that $|M(z,1^t)| = p(\ell)$.

   The first two requirements imply that $|M(z,\alpha)|$ is a function of $|z|$ and $|\alpha|$, which increases with $|\alpha|$. The third requirement implies that, for every ℓ, each string of length at most ℓ can be mapped to a string of length $p(\ell)$.

Note that Condition 1 is reproduced from Exercise 2.30, whereas Conditions 2 and 3 are new. Prove that if the set S is Karp-reducible to the set T and T is monotonically markable then S is Karp-reducible to T by a reduction that is monotone and length-regular (i.e., the reduction satisfies the conditions of Proposition 10.18).

Guideline: Given a Karp-reduction f from S to T, first obtain a length-regular reduction f' from S to T (by applying the marking algorithm to $f(x)$, while using Conditions 1 and 3c). In particular, one can guarantee that if $|x'| > |x''|$ then $|f'(x')| > |f'(x'')|$. Next, obtain a reduction f'' that is also monotone (e.g., by letting $f''(x) = M(f'(x), x)$, while using Conditions 1 and 2).[32]

Exercise 10.22 (monotone markability and markability) Prove that if a set is monotonically markable (as per Exercise 10.21) then it is markable (as per Exercise 2.30).

Guideline: Let M denote the guaranteed monotone-marking algorithm. For starters, assume that M is 1-1, and define $M'(z, \alpha) = M(z, \langle z, \alpha \rangle)$. Note that the preimage (z, α) can be found by conducting a binary search (for each of the possible values of $|z|$). In the general case, we modify the construction so that to guarantee that M' is 1-1. Specifically, let $\mathrm{idx}(n, m) = n + \sum_{i=2}^{n+m}(i-1)$ be the index of (n, m) in an enumeration of all pairs of positive integers, and p be as in Condition 3c. Then, let $M'(z, \alpha) = M(z, C_{t(|z|,|\alpha|)}(\langle z, \alpha \rangle))$, where $t(n, m) = \omega(n+m)$ satisfies $|M(1^n, 1^{t(n,m)})| = p(\mathrm{idx}(n, m))$ and $C_t(y)$ is a monotone encoding of y using a t-bit long string.

Exercise 10.23 (some monotonically markable sets) Referring to Exercise 10.21, verify that each of the twenty-one NP-complete problems treated in in Karp's first paper on NP-completeness [136] is monotonically markable. For starters, consider the sets `SAT`, `Clique`, and `3-Colorability`.

Guideline: For `SAT` consider the following marking algorithm M. This algorithm uses two (fixed) satisfiable formulae of the same length, denoted ψ_0 and ψ_1, such that $\psi_0 < \psi_1$. For any formula ϕ and any binary string $\sigma_1 \cdots \sigma_m \in \{0,1\}^m$, it holds that $M(\phi, \sigma_1 \cdots \sigma_m) = \psi_{\sigma_1} \wedge \cdots \wedge \psi_{\sigma_m} \wedge \phi$, where ψ_0 and ψ_1 use variables that do not appear in ϕ. Note that the multiple occurrences of ψ_σ can be easily avoided (by using "variations" of ψ_σ).

Exercise 10.24 (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 $(S', X') \in \mathrm{tpc}\mathcal{BPP}$, then (S, X) is in $\mathrm{tpc}\mathcal{BPP}$.

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).

[32] Actually, Condition 2 (combined with the length regularity of f') only takes care of monotonicity with respect to strings of equal length. To guarantee monotonicity with respect to strings of different length, we also use Condition 3b (and $|f'(x')| > |f'(x'')|$ for $|x'| > |x''|$).

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

Guideline: See Exercise 10.14.

Exercise 10.26 (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 $\#R(x) = 2^{|x|+p(|x|)} \cdot \Pr[A(1^{|x|+1}) = x1]$.

Exercise 10.27 (distributional versions of NPC problems – Part 1 [29]) Prove that if $S_{\mathbf{u}}$ is Karp-reducible to S by a mapping that does not shrink the input then there exists a polynomial-time sampleable ensemble X such that any problem in dist$\mathcal{NP}$ is reducible to (S,X).

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 10.25).) 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 10.28 (distributional versions of NPC problems – Part 2 [29]) Prove Theorem 10.25 (i.e., if $S_{\mathbf{u}}$ is Karp-reducible to S by a mapping 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)).

Guideline: We establish the claim for $S = S_{\mathbf{u}}$, and the general claim follows by using the reduction of $S_{\mathbf{u}}$ to S (as in Exercise 10.27). Thus, we focus on showing that, for some (suitably chosen) sampleable ensemble X, any $(S', X') \in$ samp$\mathcal{NP}$ is reducible to $(S_{\mathbf{u}}, X)$. Loosely speaking, X will be an adequate convex combination of all sampleable distributions (and thus X will neither equal U' nor be simple). Specifically, $X = \{X_n\}_{n\in\mathbb{N}}$ is defined such that the sampler for 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,[33] and outputs whatever

[33] 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.

the latter has output (or 0^n in case the said algorithm has not halted within n^3 steps). Prove that, for any $(S'', X'') \in \mathrm{samp}\mathcal{NP}$ 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).[34] Finally, using adequate padding, reduce any $(S', X') \in \mathrm{samp}\mathcal{NP}$ to some $(S'', X'') \in \mathrm{samp}\mathcal{NP}$ such that X'' is sampleable in cubic time.

Exercise 10.29 (search vs decision in the context of sampleable ensembles) Prove that every problem in $\mathrm{samp}\mathcal{NP}$ is reducible to some problem in $\mathrm{samp}\mathcal{PC}$, and every problem in $\mathrm{samp}\mathcal{PC}$ is *randomly* reducible to some problem in $\mathrm{samp}\mathcal{NP}$.

Guideline: See proof of Theorem 10.23.

[34]Note that applying this reduction to X'' yields an ensemble 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.

Epilogue

Farewell, Hans – whether you live or end where you are! Your chances are not good. The wicked dance in which you are caught up will last a few more sinful years, and we would not wager much that you will come out whole. To be honest, we are not really bothered about leaving the question open. Adventures in the flesh and spirit, which enhanced and heightened your ordinariness, allowed you to survive in the spirit what you probably will not survive in the flesh. There were majestic moments when you saw the intimation of a dream of love rising up out of death and the carnal body. Will love someday rise up out of this world-wide festival of death, this ugly rutting fever that inflames the rainy evening sky all round?

Thomas Mann, The Magic Mountain, The Thunderbolt.

We hope that this work has succeeded in conveying the fascinating flavor of the concepts, results and open problems that dominate the field of computational complexity. We believe that the new century will witness even more exciting developments in this field, and urge the reader to try to contribute to them. But before bidding goodbye, we wish to express a few more thoughts.

As noted in Section 1.1.1, so far complexity theory has been far more successful in relating fundamental computational phenomena than in providing definite answers regarding fundamental questions. Consider, for example, the theory of NP-completeness versus the P-versus-NP Question, or the theory of pseudorandomness versus establishing the existence of one-way function (even under $\mathcal{P} \neq \mathcal{NP}$). The failure to resolve questions of the "absolute" type is the source of common frustration and one often wonders about the reasons for this failure.

Our feeling is that many of these failures are really due to the difficulty of the questions asked, and that one tends to underestimate their hardness because they are so appealing and natural. Indeed, the underlying sentiment is that if a question is appealing and natural then answering it should not be hard. We doubt this sentiment. Our own feeling is that the more intuitive a question is, the harder it may be to answer. Our view is that intuitive questions arise from an encounter with the raw and chaotic reality of life, rather than from an artificial construct which is typically endowed with a rich internal structure. Indeed, natural complexity classes and natural questions regarding computation arise from looking at the reality of computation from the outside and thus lack any internal structure. Specifically, complexity classes are defined in terms of the "external behavior" of potential algorithms (i.e., the resources such algorithms require) rather than in terms of the "internal structure" (of the problem). In our opinion, this "external nature" of the definitions of complexity theoretic questions makes them hard to resolve.

Another hard aspect regarding the "absolute" (or "lower-bound") type of questions is the fact that they call for impossibility results. That is, the natural formulation of these questions calls for proving the non-existence of something (i.e., the non-existence of efficient procedures for solving the problem in question). Needless to say, proving the non-existence of certain objects is *typically* harder than proving existence of related objects (indeed, see Section 9.1). Still, proofs of non-existence of certain objects are known in various fields and in particular in complexity theory, but such proofs tend to either be trivial (see, e.g., Section 4.1) or are derived by exhibiting a sophisticated process that transforms the original question to a trivial one. Indeed, the latter case is the one that underlies many of the impressive successes of circuit complexity, and all relative results of the "high-level" direction have a similar nature (i.e., of relating one computational question to another). Thus, we are not suggesting that the "absolute" questions of complexity theory cannot be resolved, but are rather suggesting an intuitive explanation to the difficulties of resolving them.

The obvious fact that difficult questions can be resolved is demonstrated by several recent results, which are mentioned in this book and "forced" us to modify earlier drafts of it. Examples include the log-space graph exploration algorithm presented in Section 5.2.4 and the alternative proof of the PCP Theorem presented in §9.3.2.3 as well as Theorem 10.19 and the brief mention of the results of [171, 240].