Randomness and Computation

Oded Goldreich
Department of Computer Science and Applied Mathematics
Weizmann Institute of Science, Rehovot, Israel.
oded.goldreich@weizmann.ac.il

February 17, 2006

Abstract

The interplay of randomness and computation is at the heart of modern Cryptography and plays a fundamental role in the design of algorithms and in the study of computation at large. Specifically, this interplay is pivotal to several intriguing notions of probabilistic proof systems (e.g., interactive proofs, zero-knowledge proofs, and probabilistically checkable proofs), is the focal of the computational approach to randomness, and is essential for various types of sub-linear time algorithms. This essay provides a brief outline of these connections.

Contents

1 Introduction

While it is safe to assume that any living adult is aware of the revolutionary impact of the computing technology on our society, we fear that few readers have a sense of the theory of computation. This contrast is not so surprising, because people seem so overwhelmed by the wonders of this technology that they do not get to wonder about the theory underlying it. Furthermore, people tend to think of computing in the concrete terms in which they have lastly encountered it rather than in general terms. Consequently, the fascinating intellectual contents of the theory of computation is rarely understood by non-specialists.

One goal of this essay is making a tiny contribution towards a possible change in this sour state of affairs, by discussing one aspect of the theory of computation: its connection to randomness. Our guess is that the suggestion that there is a connection between computation and randomness may meet the skepticism of some readers, because computation seems the ultimate manifestation of determinism.

To address this skepticism, we suggest considering what happens when a deterministic machine (or any deterministic process) is fed with a random input or just with an input that looks random. Indeed, one contribution of the theory of computation (further discussed in Section 2) is a definition of "objects that look random" (a notion which makes sense even if the real world is actually deterministic).

Still one may wonder whether we can obtain or generate objects that look random. For example, can we toss a coin (in the sense that one cannot feasibly predict the answer before seeing it)? Assuming a positive answer, we may also assume that unpredictable values can be obtained by other mechanical and/or electrical processes, which suggest that computers can also obtain such values. The question then is what benefit can be achieved by using such random (or unpredictable) values.

A major application of random (or unpredictable) values is to the area of Cryptography (see Section 4). In fact, the very notion of a secret refers to such a random (or unpredictable) value. Furthermore, various natural security concerns (e.g., private communication) can be met by employing procedures that make essential use of such secrets and/or random values.

Another major application of random (or unpredictable) values is to various sampling procedures. In Section 5, we consider a wider perspective on such procedures, viewing them as a special type of super fast procedures called *sub-linear time algorithms*. Such a procedure cannot afford to scan the entire input, but rather probes few (randomly) selected locations in it and, based on these few values, attempts to make a meaningful assertion regarding the entire input. Indeed, we assume that the reader is aware of the fact that random sampling allows to approximate the fraction of the population that votes for a particular candidate. Our point is that other global properties of the input, which are not merely averages of various types, can also be approximated by sampling.

Lastly, we mention that randomized verification procedures yield fascinating types of probabilistic proof systems, which are discussed in Section 3. In particular, such proof systems demonstrate the advantage of interaction (over one-directional communication) and the possibility of decoupling proving from learning (i.e., the possibility of proving an assertion without yielding anything beyond its validity). Other forms of probabilistic proof systems allow for super fast verification (based on probing few locations in a redundant proof, indeed as in the aforementioned sublinear-time algorithms).

Before discussing the foregoing applications of randomness in greater length, we provide a somewhat wider perspective on the theory of computation as well as present some of its central conventions. We will also clarify what randomness means in that theory (and in this article).

1.1 A wider perspective on the theory of computation

The theory of computation aims at understanding general properties of computation be it natural, man-made, or imaginary. Most importantly, it aims to understand the nature of efficient computation. We demonstrate these issues by briefly considering a few typical questions.

A key question is *which functions can be efficiently computed?* For example, it is (relatively) easy to multiply integers, but it seems hard to take the product and factor it into its prime components. In general, it seems that there are one-way computations, or put differently *one-way functions*: Such functions are easy to evaluate but hard to invert (even in an average-case sense). But do one-way functions exist? It is widely believed that the answer is positive, and this question is related to other fundamental questions.

A related question is that of the comparable difficulty of *solving problems versus verifying the correctness of solutions*. Indeed our daily experience is that it is harder to solve a problem than it is to check the correctness of a solution (e.g., think of either a puzzle or a research problem). Is this experience merely a coincidence or does it represent a fundamental fact of life (or a property of the world)? Could you imagine a world in which solving any problem is not significantly harder than checking a solution to it? Would the term "solving a problem" not lose its meaning in such a hypothetical (and impossible in our opinion) world? The denial of the plausibility of such a hypothetical world (in which "solving" is not harder than "checking") is what the celebrated "P different from NP" conjecture means, where P represents tasks that are efficiently solvable and NP represents tasks for which solutions can be efficiently checked for correctness.

The theory of computation is also concerned with finding the most efficient methods for solving specific problems. To demonstrate this line of research we mention that the simple (and standard) method for multiplying numbers that is taught in elementary school is not the most efficient one possible. Multiplying two n-digit long numbers by this method requires n^2 single-digit multiplications (and a similar number of single-digit additions). In contrast, consider writing these numbers as $10^{n/2} \cdot a' + a''$ and $10^{n/2} \cdot b' + b''$, where a', a'', b', b'' are $n/2$-digit long numbers, and note that

$$(10^{n/2} \cdot a' + a'') \times (10^{n/2} \cdot b' + b'') = 10^n \cdot P_1 + 10^{n/2} \cdot (P_2 - P_1 - P_3) + P_3$$
$$\text{where } P_1 = a' \times b',\ P_2 = (a' + a'') \times (b' + b''), \text{ and } P_3 = a'' \times b''.$$

Thus, multiplying two n-digit long numbers requires only three (rather than four) multiplications of $n/2$-digit long numbers (and a constant number of additions of $n/2$-digit long numbers and "shifts" of n-digit long numbers (indicated by $\cdot$)). Letting $M(n)$ denote the complexity of multiplying two n-digit long numbers, we obtain $M(n) < 3 \cdot M(n/2) + c \cdot n$, where c is some constant (independent of n), which solves to $M(n) < c' \cdot 3^{\log_2 n} = c' \cdot n^{\log_2 3} < n^{1.6}$ (for some constant c'). We mention that this is not the best known algorithm; the latter runs in time $\mathrm{poly}(\log n) \cdot n$.

The theory of computation provides a new viewpoint on old phenomena. We have already mentioned the computational approaches to randomness (see Section 2) and to proofs, interaction, knowledge, and learning (see Section 3). Additional natural concepts given an appealing computational interpretations include the *importance of representation*, the notion of *explicitness*, and the possibility that approximation is easier than optimization (see Section 5). Let us say a few words about representation and explicitness.

The foregoing examples hint to *the importance of representation*, because in all these computational problems the solution is implicit in the problem's statement. That is, the problem contains all necessary information, and one merely needs to process this information in order to supply the answer.[1] Thus, the theory of computation is concerned with the manipulation of information, and its transformation from one representation (in which the information is given) to another representation (which is the one desired). Indeed, a solution to a computational problem is merely a different representation of the information given; that is, a representation in which the answer is explicit rather than implicit. For example, the answer to the question of whether or not a given system of quadratic equations has an integer solution is implicit in the system itself (but the task is to make the answer explicit). Thus, the theory of computation clarifies a central issue regarding representation; that is, the distinction between what is explicit and what is implicit in a representation. Furthermore, it also suggests a quantification of the level of non-explicitness.

1.2 Important conventions for the theory of computation

In light of the foregoing discussion it is important to specify the representation used in computational problems. Actually, a computational problem refer to an infinite set of *finite objects*, called the problem's instances, and specifies the desired solution for each instance. For example, the instances of the `multiplication problem` are pairs of natural numbers, and the desired solution is the corresponding product. Objects are represented by finite binary sequences, called strings.[2] For a natural number n, we denote by $\{0,1\}^n$ the set of all strings of length n, hereafter referred to as n-bit strings. The set of all strings is denoted $\{0,1\}^*$; that is, $\{0,1\}^* = \cup_{n\in\mathbb{N}}\{0,1\}^n$.

Another piece of terminology is the term algorithm, which refers to an automated procedure designed to solve some computational task. A rigorous definition requires specifying a reasonable model of computation, but the specifics of this model are not important for the current essay. We focus on efficient algorithms, which are commonly defined as making a number of steps that is polynomial in the length of their input.[3] Indeed, asymptotic analysis (or rather a functional treatment of the running time of algorithms in terms of the length of their input) is a central convention in the theory

[1] In contrast, in other disciplines, solving a problem may also require gathering information that is not available in the problem's statement. This information may either be available from auxiliary (past) records or be obtained by conducting new experiments.

[2] Indeed, in the foregoing example, we used the daily representation of numbers as sequences of decimal digits, but in the theory of computation natural numbers are typically represented by their binary expansion.

[3] In Section 5 we consider even faster algorithms, which make (significantly) less steps than the length of their input, but such algorithms can only provide approximate solutions.

of computation.[4]

Typically, our notion of efficient algorithms will include also *probabilistic* (polynomial-time) algorithms; that is, algorithms that can "toss coins" (i.e., make random choices). For each reasonable model of computation, probabilistic (or randomized) algorithms are defined as standard algorithm augmented with the ability to choose uniformly among a finite number of (say two) predetermined possibilities. That is, at each computation step, such an algorithm makes a move that is chosen uniformly among two predetermined possibilities.

1.3 Randomness in the context of computation

Throughout the entire essay we will refer only to *discrete* probability distributions. The support of such distributions will be associated with a set of strings, typically of the same length.

For the purpose of asymptotic analysis, we will often consider probability ensembles, which are sequences of distributions that are indexed either by integers or by strings. For example, throughout the essay, we let $\{U_n\}_{n\in\mathbb{N}}$ denote the uniform ensemble, where U_n is uniform over the set of strings of length n; that is, $\mathsf{Pr}_{z\sim U_n}[z=\alpha]$ equals 2^{-n} if $\alpha \in \{0,1\}^n$ and equals 0 otherwise. We will often denote by $\{D_n\}_{n\in\mathbb{N}}$ (or $\{D_s\}_{s\in S}$, where $S \subseteq \{0,1\}^*$) a generic probability ensemble, and typically it will be the case that there exists some function $\ell : \mathbb{N}\to\mathbb{N}$ such that $\mathsf{Pr}_{z\sim D_n}[z \in \{0,1\}^{\ell(n)}] = 1$ (resp., $\mathsf{Pr}_{z\sim D_s}[z \in \{0,1\}^{\ell(n)}] = 1$, where n denotes the length of s). Furthermore, typically, ℓ will be a polynomial.

One important case of probability ensembles is that of ensembles that represent the output of randomized processes (e.g., randomized algorithms). Letting $A(x)$ denote the output of the probabilistic (or randomized) algorithm A on input x, we may consider the probability ensemble $\{A(x)\}_{x\in\{0,1\}^*}$. Indeed, if A is a probabilistic polynomial-time algorithm then $A(x)$ is distributed over strings of length that is bounded by a polynomial in the length of x.

On the other hand, we say that a probability ensemble $\{D_n\}_{n\in\mathbb{N}}$ (resp., $\{D_s\}_{s\in S}$) is efficiently sampleable if there exists a probabilistic polynomial-time algorithm A such that for every $n \in \mathbb{N}$ it holds that $A(1^n) \equiv D_n$ (resp.,

[4] We stress, however, that asymptotic (or functional) treatment is not essential to this theory, but rather provides a convenient framework. One may develop the entire theory in terms of inputs of fixed lengths and concrete bounds on the number of steps taken by corresponding algorithms. However, such an alternative treatment is more cumbersome.

for every $s \in S$ it holds that $A(s) \equiv D_s$). That is, algorithm A makes a number of steps that is polynomial in n, and produces a sample distributed according to D_n (resp., D_s, where n denotes the length of s).

We will often talk of "random bits" and mean values selected uniformly and independently in $\{0,1\}$. In particular, randomized algorithms may be viewed as deterministic algorithms that are given an adequate number of random bits as an auxiliary input. This means that rather than viewing these algorithms as making random choices, we view them as determining these choices according to a sequence of random bits that is generated by some outside process.

1.4 The rest of this essay

In the rest of this essay we briefly review the theory of pseudorandomness (Section 2), three types of probabilistic proof systems (Section 3), the theoretical foundations of Cryptography (Section 4), and sublinear-time algorithms (Section 5). Needless to say, these overviews are the tip of an iceberg, and the interested reader will be referred to related texts for further information. In general, the most relevant text is [7], which provides more extensive overviews of the first three areas.

In addition, we recommend textbooks such as [20, 24] for background on the aspects of the theory of computation that are most relevant for the current essay. We note that randomized algorithms and procedures are valuable also in settings not discussed in the current essay (e.g., for polynomial-time computations as well as in the context of distributed and parallel computation). The interested reader is referred to [19].

An apology. Our feeling is that in an essay written for a general readership it makes no sense to provide the standard scholarly citations. The most valuable references for such readers are relevant textbooks and expository articles, written with the intension of communicating to non-experts. On the other hand, the general reader may be interested in having some sense of the history of the field, and thus references to few pioneering works seem adequate. We are aware that in trying to accommodate the non-experts, we may annoy the experts, and hence the current apology to all experts who made an indispensable contribution to the development of these areas and who's work was victim to our referencing policy.

2 Pseudorandomness

Indistinguishable things are identical.[5]

G.W. Leibniz (1646–1714)

A fresh view at the *question of randomness* has been taken in the theory of computation: It has been postulated that a distribution is pseudorandom if it cannot be told apart from the uniform distribution by any efficient procedure. The paradigm, originally associating efficient procedures with polynomial-time algorithms, has been applied also with respect to a variety of limited classes of such distinguishing procedures.

At the extreme, this approach says that the question of whether the world is deterministic or allows for some free choice (which may be viewed as sources of randomness) is irrelevant. *What matters is how the world looks to us and to various computationally bounded devices.* That is, if some phenomenon looks random then we may just treat it as if it were random. Likewise, if we can generate sequences that cannot be told apart from the uniform distribution by any efficient procedure, then we can use these sequences in any efficient randomized application instead of the ideal random bits that are postulated in the design of this application.

2.1 A wider context and an illustration

The second half of this century has witnessed the development of three theories of randomness, a notion which has been puzzling thinkers for ages. The first theory (cf., [4]), initiated by Shannon, is rooted in probability theory and is focused at distributions that are not perfectly random (i.e., are not uniform over a set of strings of adequate length). Shannon's Information Theory characterizes perfect randomness as the extreme case in which the *information contents* is maximized (i.e., the strings contain no redundancy at all). Thus, perfect randomness is associated with a unique distribution: the uniform one. In particular, by definition, one cannot (deterministically) generate such perfect random strings from shorter random seeds.

The second theory (cf., [17]), initiated by Solomonov, Kolmogorov, and Chaitin, is rooted in computability theory and specifically in the notion of

[5]This is the *Principle of Identity of Indiscernibles.* Leibniz admits that counterexamples to this principle are conceivable but will not occur in real life because God is much too benevolent. We thus believe that he would have agreed to the theme of this section, which asserts that *indistinguishable things should be considered as identical.*

a universal language (equiv., universal machine or computing device). It measures the complexity of objects in terms of the shortest program (for a fixed universal machine) that generates the object. Like Shannon's theory, Kolmogorov Complexity is quantitative and perfect random objects appear as an extreme case. However, in this approach one may say that a single object, rather than a distribution over objects, is perfectly random. Still, Kolmogorov's approach is inherently intractable (i.e., Kolmogorov Complexity is uncomputable), and – by definition – one cannot (deterministically) generate strings of high Kolmogorov Complexity from short random seeds.

The third theory, initiated by Blum, Goldwasser, Micali and Yao [13, 3, 25], is rooted in the notion of efficient computations and is the focus of this section. This approach is explicitly aimed at providing a notion of randomness that nevertheless allows for an efficient generation of random strings from shorter random seeds. The heart of this approach is the suggestion to view objects as equal if they cannot be told apart by any efficient procedure. Consequently, a distribution that cannot be efficiently distinguished from the uniform distribution will be considered as being random (or rather called pseudorandom). Thus, randomness is not an "inherent" property of objects (or distributions) but is rather relative to an observer (and its computational abilities). To demonstrate this approach, let us consider the following mental experiment.

> Alice and Bob play "head or tail" in one of the following four ways. In each of them Alice flips an unbiased coin and Bob is asked to guess its outcome *before* the coin hits the floor. The alternative ways differ by the knowledge Bob has before making his guess.
>
> In the first alternative, Bob has to announce his guess before Alice flips the coin. Clearly, in this case Bob wins with probability 1/2.
>
> In the second alternative, Bob has to announce his guess while the coin is spinning in the air. Although the outcome is *determined in principle* by the motion of the coin, Bob does not have accurate information on the motion and thus we believe that also in this case Bob wins with probability 1/2.
>
> The third alternative is similar to the second, except that Bob has at his disposal sophisticated equipment capable of providing accurate *information* on the coin's motion as well as on the en-

vironment effecting the outcome. However, Bob cannot process this information in time to improve his guess.

In the fourth alternative, Bob's recording equipment is directly connected to a *powerful computer* programmed to solve the motion equations and output a prediction. It is conceivable that in such a case Bob can improve substantially his guess of the outcome of the coin.

We conclude that the randomness of an event is relative to the information and computing resources at our disposal. Thus, a natural concept of pseudorandomness arises: a distribution is *pseudorandom* if no efficient procedure can distinguish it from the uniform distribution, where efficient procedures are associated with (probabilistic) polynomial-time algorithms. This notion of pseudorandomness is indeed the most fundamental one, and the current section is focused on it.[6]

The foregoing discussion has focused at one aspect of the pseudorandomness question: the resources or type of the observer (or potential distinguisher). Another important aspect is whether such pseudorandom sequences can be generated from much shorter ones, and at what cost (i.e., at what computational effort). A natural approach is that the generation process has to be at least as efficient as the distinguisher (equiv., that the distinguisher is allowed at least as much resources as the generator). Coupled with the aforementioned strong notion of pseudorandomness, this yields the archetypical notion of pseudorandom generators – these operating in polynomial-time and producing sequences that are indistinguishable from uniform ones by *any* polynomial-time observer. Such (general-purpose) pseudorandom generators enable reducing the randomness complexity of *any efficient application*, and are thus of great relevance to randomized algorithms and Cryptography (see Sections 2.5 and 4). Indeed, these general-purpose pseudorandom generators will be the focus of the current section.[7] Fur-

[6]We mention that weaker notions of pseudorandomness arise as well; they refer to indistinguishability by weaker procedures such as space-bounded algorithms (see [7, Sec. 3.5]), constant-depth circuits, etc. Stretching this approach even further one may consider algorithms that are designed on purpose so not to distinguish even weaker forms of "pseudorandom" sequences from random ones (such algorithms arise naturally when trying to convert some natural randomized algorithm into deterministic ones; see [7, Sec. 3.6]).

[7]We mention that there are important reasons for considering also an alternative that seems less natural; that is, allowing the pseudorandom generator to use more resources (e.g., time or space) than the observer it tries to fool. This alternative is natural in the context of derandomization (i.e., converting randomized algorithms to deterministic

ther discussion of the conceptual contents of this approach is provided in Section 2.6.

2.2 The notion of pseudorandom generators

Loosely speaking, a pseudorandom generator is an *efficient* program (or algorithm) that *stretches* short random strings into long *pseudorandom* sequences. We stress that the generator itself is deterministic and that the randomness involved in the generation process is captured by its input. We emphasize three fundamental aspects in the notion of a pseudorandom generator:

1. Efficiency. The generator has to be efficient. Since we associate efficient computations with polynomial-time ones, we postulate that the generator has to be implementable by a deterministic polynomial-time algorithm.

   This algorithm takes as input a string, called its seed. The seed captures a bounded amount of randomness used by a device that "generates pseudorandom sequences." The formulation views any such device as consisting of a deterministic procedure applied to a random seed.

2. Stretching. The generator is required to stretch its input seed to a longer output sequence. Specifically, it stretches n-bit long seeds into $\ell(n)$-bit long outputs, where $\ell(n) > n$. The function ℓ is called the stretching measure (or stretching function) of the generator.

3. Pseudorandomness. The generator's output has to look random to any efficient observer. That is, any efficient procedure should fail to distinguish the output of a generator (on a random seed) from a truly random bit-sequence of the same length. The formulation of the last sentence refers to a general notion of computational indistinguishability that is the heart of the entire approach.

To demonstrate the foregoing, consider the following suggestion for a pseudorandom generator. The seed consists of a pair of 500-bit integers, denoted x and N, and a million-bit long output is obtained by repeatedly squaring

ones), where the crucial step is replacing the "random source" of a fixed algorithm by a pseudorandom source, which in turn can be deterministically emulated based on a much shorter random source. For further clarification and demonstration of the usefulness of this approach the interested reader is referred to [7, Sec. 3.4&3.5].

the current x modulo N and emitting the least significant bit of each intermediate result (i.e., let $x_i \leftarrow x_{i-1}^2 \bmod N$, for $i = 1, ..., 10^6$, and output $b_1, b_2, ..., b_{10^6}$, where $x_0 \stackrel{\text{def}}{=} x$ and b_i is the least significant bit of x_i). This process may be generalized to seeds of length n (here we used $n = 1000$) and outputs of length $\ell(n)$ (here $\ell(1000) = 10^6$). Such a process certainly satisfies Items (1) and (2) above, whereas the question whether Item (3) holds is debatable (once a rigorous definition is provided). As a special case of Theorem 2.6 (which follows), we mention that, under the assumption that it is difficult to factor large integers, a slight variant of the foregoing process is indeed a pseudorandom generator.

Computational indistinguishability. Intuitively, two objects are called computationally indistinguishable if no efficient procedure can tell them apart. Here the objects are (fixed) probability distributions (or rather ensembles), and the observer is given a sample drawn from one of the two distributions and is asked to tell from which distribution it was taken (e.g., it is asked to say "1" if the sample is taken from the first distribution). Following the asymptotic framework (see Sections 1.2 and 1.3), the foregoing discussion is formalized as follows.

Definition 2.1 (computational indistinguishability [13, 25]). *Two probability ensembles, $\{X_n\}_{n\in\mathbb{N}}$ and $\{Y_n\}_{n\in\mathbb{N}}$, are called* computationally indistinguishable *if for any probabilistic polynomial-time algorithm A, any positive polynomial p, and all sufficiently large n*

$$\left| \mathsf{Pr}_{x\sim X_n}[A(x) = 1] \; - \; \mathsf{Pr}_{y\sim Y_n}[A(y) = 1] \right| \; < \; \frac{1}{p(n)} \, . \qquad (1)$$

The probability is taken over X_n (resp., Y_n) as well as over the internal coin tosses of algorithm A.

Algorithm A, which is called a potential distinguisher, is given a sample (which is drawn either from X_n or from Y_n) and its output is viewed as an attempt to tell whether this sample was drawn from X_n or from Y_n. Eq. (1) requires that such an attempt is bound to fail; that is, the outcome 1 (possibly representing a verdict that the sample was drawn from X_n) is essentially as likely to occur when the sample is drawn from X_n as when it is drawn from Y_n.

A few comments are in order. Firstly, the distinguisher (i.e., A) is allowed to be probabilistic. This makes the requirement only stronger, and seems essential to several important aspects of our approach. Secondly, we view events occuring with probability that is upper bounded by the reciprocal of polynomials as negligible (e.g., $2^{-\sqrt{n}}$ is negligible as a function of n). This is well-coupled with our notion of efficiency (i.e., polynomial-time computations): an event that occurs with negligible probability (as a function of a parameter n), will also occur with negligible probability if the experiment is repeated for poly(n)-many times. Thirdly, for efficiently sampleable ensembles, computational indistinguishability is preserved also when providing the distinguisher with polynomially many samples (of the tested distribution). Lastly we note that computational indistinguishability is a coarsening of statistical indistinguishability; that is, waiving the computational restriction on the distinguisher is equivalent to requiring that the variation distance between X_n and Y_n (i.e., $\sum_z |X_n(z) - Y_n(z)|$) is negligible (in n).

An important case in which computational indistinguishability is strictly more liberal than statistical indistinguishability arises from the notion of a pseudorandom generator.

Definition 2.2 (pseudorandom generators [3, 25]). *A deterministic polynomial-time algorithm G is called a* pseudorandom generator *if there exists a* stretching function, $\ell:\mathbb{N}\to\mathbb{N}$ (i.e., $\ell(n) > n$), *such that the following two probability ensembles, denoted $\{G_n\}_{n\in\mathbb{N}}$ and $\{R_n\}_{n\in\mathbb{N}}$, are computationally indistinguishable.*

1. *Distribution G_n is defined as the output of G on a uniformly selected seed in $\{0,1\}^n$.*
2. *Distribution R_n is defined as the uniform distribution on $\{0,1\}^{\ell(n)}$.*

Note that $G_n \equiv G(U_n)$, whereas $R_n = U_{\ell(n)}$. Requiring that these two ensembles are computationally indistinguishable means that, for any probabilistic polynomial-time algorithm A, the detected (by A) difference between G_n and R_n, denoted

$$d_A(n) \stackrel{\text{def}}{=} \left| \mathsf{Pr}_{s\sim U_n}[A(G(s)) = 1] \; - \; \mathsf{Pr}_{r\sim U_{\ell(n)}}[A(r) = 1] \right|$$

is negligible (i.e., $d_A(n)$ vanishes faster than the reciprocal of any polynomial). Thus, pseudorandom generators are efficient (i.e., polynomial-time) deterministic programs that expand short randomly selected seeds into longer pseudorandom bit sequences, where the latter are defined as computationally indistinguishable from truly random bit-sequences by efficient (i.e., polynomial-time) algorithms. It follows that any efficient randomized algorithm maintains its performance when its internal coin tosses are substituted by a sequence generated by a pseudorandom generator. That is:

Construction 2.3 (typical application of pseudorandom generators). *Let A be a probabilistic polynomial-time algorithm, and $\rho(n)$ denote an upper bound on the number of coins that A tosses on n-bit inputs* (e.g., $\rho(n) = n^2$). *Let $A(x,r)$ denote the output of A on input x and coin tossing sequence $r \in \{0,1\}^{\rho(n)}$, where n denotes the length of x. Let G be a pseudorandom generator with stretching function $\ell:\mathbb{N}\to\mathbb{N}$* (e.g., $\ell(k) = k^5$). *Then A_G is a randomized algorithm that on input $x \in \{0,1\}^n$, proceeds as follows. It sets $k = k(n)$ to be the smallest integer such that $\ell(k) \geq \rho(n)$* (e.g., $k^5 \geq n^2$), *uniformly selects $s \in \{0,1\}^k$, and outputs $A(x,r)$, where r is the $\rho(|x|)$-bit long prefix of $G(s)$.*

Thus, using A_G instead of A, the number of random bits used by the algorithm is reduced from ρ to $\ell^{-1} \circ \rho$ (e.g., from n^2 to $k(n) = n^{2/5}$), while it is infeasible to find inputs (i.e., x's) on which the *noticeable behavior* of A_G is different from the one of A. That is, we save randomness while maintaining performance (see Section 2.5).

Amplifying the stretch function. Pseudorandom generators as in Definition 2.2 are only required to stretch their input a bit; for example, stretching n-bit long inputs to $(n+1)$-bit long outputs will do. Clearly, generators with such moderate stretch functions are of little use in practice. In contrast, we want to have pseudorandom generators with an arbitrary long stretch function. By the efficiency requirement, the stretch function can be at most polynomial. It turns out that pseudorandom generators with the smallest possible stretch function can be used to construct pseudorandom generators with any desirable polynomial stretch function. That is:

Theorem 2.4 [8, Sec. 3.3.2]. *Let G be a pseudorandom generator with stretch function $\ell(n) = n+1$, and ℓ' be any positive polynomial such that $\ell'(n) \geq n+1$. Then there exists a pseudorandom generator with stretch function ℓ'. Furthermore, the construction of the latter consists of invoking G for ℓ' times.*

Thus, when talking about the existence of pseudorandom generators, we may ignore the stretch function.

2.3 How to Construct Pseudorandom Generators

The known constructions of pseudorandomness generators are based on one-way functions. Loosely speaking, a *polynomial-time computable* function is called one-way if any efficient algorithm can invert it only with negligible success probability. For simplicity, we consider only length-preserving one-way functions.

Definition 2.5 (one-way function). *A* one-way function*, f, is a polynomial-time computable function such that for every probabilistic polynomial-time algorithm A', every positive polynomial $p(\cdot)$, and all sufficiently large n*

$$\mathsf{Pr}_{x\sim U_n}\left[A'(f(x))\in f^{-1}(f(x))\right] \;<\; \frac{1}{p(n)}\,,$$

where $f^{-1}(y) = \{z : f(z)=y\}$.

It is widely believed that one-way functions exists. Popular candidates for one-way functions are based on the conjectured intractability of integer factorization, the discrete logarithm problem, and decoding of random linear code. Assuming that integer factorization is indeed infeasible, one can prove that a minor modification of the construction outlined at the beginning of Section 2.2 constitutes a pseudorandom generator. More generally, it turns out that pseudorandom generators can be constructed based on any one-way function.

Theorem 2.6 (existence of pseudorandom generators [15]). *Pseudorandom generators exist if and only if one-way functions exist.*

To show that the existence of pseudorandom generators implies the existence of one-way functions, consider a pseudorandom generator G with stretch function $\ell(n) = 2n$. For $x, y \in \{0,1\}^n$, define $f(x,y) \stackrel{\text{def}}{=} G(x)$, so that f is polynomial-time computable (and length-preserving). It must be that f is one-way, or else one can distinguish $G(U_n)$ from U_{2n} by trying to invert and checking the result: Inverting f on its range distribution refers to the distribution $G(U_n)$, whereas the probability that U_{2n} has inverse under f is negligible. The interesting direction is the construction of pseudorandom generators based on any one-way function. A treatment of some natural special cases is provided in [8, Sec. 3.4-3.5].

2.4 Pseudorandom Functions

Pseudorandom generators allow one to efficiently generate long pseudorandom sequences from short random seeds (e.g., using n random bits, we can efficiently generate a pseudorandom bit-sequence of length n^2). Pseudorandom functions (defined below) are even more powerful: they allow efficient direct access to a huge pseudorandom sequence (which is infeasible to scan bit-by-bit). For example, based on n random bits, we define a sequence of length 2^n such that we can efficiently retrieve any desired bit in this sequence while the retrieved bits look random. In other words, pseudorandom functions can replace truly random functions in any efficient application (e.g., most notably in Cryptography). That is, pseudorandom functions are indistinguishable from random functions by any efficient procedure that may obtain the function values at arguments of its choice. Such procedures are called oracle machines, and if M is such machine and f is a function, then $M^f(x)$ denotes the computation of M on input x when M's queries are answered by the function f (i.e., during its computation M generates special strings called queries such that in response to the query q machine M is given the value $f(q)$).

Definition 2.7 (pseudorandom functions [10]). *A* pseudorandom function (ensemble)*, with length parameters $\ell_{\rm D}, \ell_{\rm R} : \mathbb{N} \to \mathbb{N}$, is a collection of functions $\{F_n\}_{n\in\mathbb{N}}$, where*

$$F_n \stackrel{\rm def}{=} \{f_s : \{0,1\}^{\ell_{\rm D}(n)} \to \{0,1\}^{\ell_{\rm R}(n)}\}_{s\in\{0,1\}^n},$$

satisfying

- (efficient evaluation). *There exists an efficient* (deterministic) *algorithm that when given a* seed*, s, and an $\ell_{\rm D}(n)$-bit* argument*, x, returns the $\ell_{\rm R}(n)$-bit long value $f_s(x)$, where n denotes the length of s.*

  (Thus, the seed s is an "effective description" of the function f_s.)

- (pseudorandomness). *For every probabilistic polynomial-time oracle machine M, every positive polynomial p, and all sufficiently large n*

  $$\left| \mathsf{Pr}_{s\sim U_n}[M^{f_s}(1^n) = 1] - \mathsf{Pr}_{\rho\sim R_n}[M^{\rho}(1^n) = 1] \right| < \frac{1}{p(n)},$$

  where R_n denotes the uniform distribution over all functions mapping $\{0,1\}^{\ell_{\rm D}(n)}$ to $\{0,1\}^{\ell_{\rm R}(n)}$.

Suppose, for simplicity, that $\ell_{\mathrm{D}}(n) = n$ and $\ell_{\mathrm{R}}(n) = 1$. Then a function uniformly selected among 2^n functions (of a pseudorandom ensemble) presents an input-output behavior indistinguishable in $\mathrm{poly}(n)$-time from the one of a function selected at random among all the 2^{2^n} Boolean functions. Contrast this with a distribution over 2^n sequences, produced by a pseudorandom generator applied to a random n-bit seed, which is computationally indistinguishable from the uniform distribution over $\{0,1\}^{\mathrm{poly}(n)}$ (which has a support of size $2^{\mathrm{poly}(n)}$). Still pseudorandom functions can be constructed from any pseudorandom generator.

Theorem 2.8 (how to construct pseudorandom functions [10]). *Let G be a pseudorandom generator with stretching function $\ell(n) = 2n$. For $s \in \{0,1\}^n$, let $G_0(s)$ (resp., $G_1(s)$) denote the first (resp., last) n bits in $G(s)$, and let*

$$G_{\sigma_n \cdots \sigma_2 \sigma_1}(s) \stackrel{\text{def}}{=} G_{\sigma_n}(\cdots G_{\sigma_2}(G_{\sigma_1}(s)) \cdots).$$

That is, $G_x(s)$ is computed by successive applications of either G_0 or G_1 to the current n-bit long string, where the decision which of the two mappings to apply is determined by the corresponding bit of x. Let $f_s(x) \stackrel{\text{def}}{=} G_x(s)$ and consider the function ensemble $\{F_n\}_{n\in\mathbb{N}}$, where $F_n = \{f_s : \{0,1\}^n \to \{0,1\}^n\}_{s\in\{0,1\}^n}$. Then this ensemble is pseudorandom (with length parameters $\ell_{\mathrm{D}}(n) = \ell_{\mathrm{R}}(n) = n$).

The foregoing construction can be easily adapted to any (polynomially-bounded) length parameters $\ell_{\mathrm{D}}, \ell_{\mathrm{R}} : \mathbb{N} \to \mathbb{N}$.

2.5 The Applicability of Pseudorandom Generators

Randomness is playing an increasingly important role in computation: it is frequently used in the design of sequential, parallel, and distributed algorithms, and is of course central to Cryptography. Whereas it is convenient to design such algorithms making free use of randomness, it is also desirable to minimize the use of randomness in real implementations since generating perfectly random bits via special hardware is quite expensive. Thus, pseudorandom generators (as in Definition 2.2) are a key ingredient in an "algorithmic tool-box": they provide an automatic compiler of programs written with free use of randomness into programs that make an economical use of randomness.

Indeed, "pseudo-random number generators" have appeared with the first computers. However, typical implementations use generators that are not pseudorandom according to Definition 2.2. Instead, at best, these generators are shown to pass *some* ad-hoc statistical test. We warn that the fact that a "pseudo-random number generator" passes some statistical tests does not mean that it will pass a new test and that it is good for a future (untested) application. Furthermore, the approach of subjecting the generator to some ad-hoc tests fails to provide general results of the type stated above (i.e., of the form "for *all* practical purposes using the output of the generator is as good as using truly unbiased coin tosses"). In contrast, the approach encompassed in Definition 2.2 aims at such generality, and in fact is tailored to obtain it: the notion of computational indistinguishability, which underlines Definition 2.2, covers all possible efficient applications postulating that for all of them pseudorandom sequences are as good as truly random ones.

Pseudorandom generators and functions are of key importance in Cryptography. In particular, they are typically used to establish private-key encryption and authentication schemes. For further discussion see Section 4.

2.6 The Intellectual Contents of Pseudorandom Generators

We shortly discuss some intellectual aspects of pseudorandom generators as defined above.

Behavioristic versus ontological. Our definition of pseudorandom generators is based on the notion of computational indistinguishability. The behavioristic nature of the latter notion is best demonstrated by confronting it with the Kolmogorov-Chaitin approach to randomness. Loosely speaking, a string is *Kolmogorov-random* if its length equals the length of the shortest program producing it. This shortest program may be considered the "true explanation" to the phenomenon described by the string. A Kolmogorov-random string is thus a string that does not have a substantially simpler (i.e., shorter) explanation than itself. Considering the simplest explanation of a phenomenon may be viewed as an ontological approach. In contrast, considering the effect of phenomena (on an observer), as underlying the definition of pseudorandomness, is a behavioristic approach. Furthermore, there exist probability distributions that are not uniform (and are not even statistically close to a uniform distribution) but nevertheless are indistinguishable from a uniform distribution by any efficient procedure. Thus, distributions that are ontologically very different are considered equivalent by the behavioristic point of view taken in the Definition 2.1.

A relativistic view of randomness. Pseudorandomness is defined in terms of its observer: It is a distribution that cannot be told apart from a uniform distribution by any efficient (i.e., polynomial-time) observer. However, pseudorandom sequences may be distinguished from random ones by infinitely powerful computers (not at our disposal!). Furthermore, a machine that runs in exponential-time can distinguish the output of a pseudorandom generator from a uniformly selected string of the same length (e.g., just by trying all possible seeds). Thus, pseudorandomness is subjective, dependent on the abilities of the observer.

Randomness and computational difficulty. Pseudorandomness and computational difficulty play dual roles: The definition of pseudorandomness relies on the fact that placing computational restrictions on the observer gives rise to distributions that are not uniform and still cannot be distinguished from uniform. Furthermore, the known constructions of pseudorandom generators relies on conjectures regarding computational difficulty (e.g., the existence of one-way functions), and this is inevitable: the existence of pseudorandom generators implies the existence of one-way functions.

Randomness and Predictability. The connection between pseudorandomness and unpredictability (by efficient procedures) plays an important role in the analysis of several constructions of pseudorandom generators (see [8, Sec. 3.3.5&3.5]). We wish to highlight the intuitive appeal of this connection.

2.7 Suggestions for further reading

A detailed textbook presentation of the material that is reviewed in this section is provided in [8, Chap. 3]. For a wider perspective, which treats this material as a special case of a general paradigm, the interested reader is referred to [7, Chap. 3].

3 Probabilistic Proof Systems

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

The notion of a verification procedure assumes the notion of computation and furthermore the notion of efficient computation. This implicit assumption is made explicit in the following definition in which efficient computation is associated with deterministic polynomial-time algorithms.

Definition 3.1 (NP-proof systems): *Let* $S \subseteq \{0,1\}^*$ *and* $\nu : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}$ *be a function such that* $x \in S$ *if and only if there exists a* $w \in \{0,1\}^*$ *that satisfies* $\nu(x,w) = 1$*. If* ν *is computable in time bounded by a polynomial in the length of its first argument then we say* ν *defines an* NP-proof system for S *and that* S *is an* NP-set*. The class of NP-sets is denoted* $\mathcal{NP}$.

Indeed, ν represents a verification procedure for claims of membership in a set S, a string w satisfying $\nu(x,w) = 1$ is a proof that x belongs to S, whereas $x \notin S$ has no such proofs. For example, consider the set of systems of quadratic equations that have integer solutions, which is a well-known NP-set. Clearly, any integer solution $\overline{v}$ to such a system Q constitutes an "NP-proof" for the assertion `the system` Q `has an integer solution` (the verification procedure consists of substituting the variables of Q by the values provided in $\overline{v}$ and computing the value of the resulting arithmetic expression).

We seize the opportunity to note that the celebrated "P different from NP" conjecture asserts that NP-proof systems are useful in the sense that *there are assertions for which obtaining a proof provides help to somebody that wishes to verify the correctness of the assertion.*[8] This conforms with our daily experience by which reading a proof eases the verification of an assertion.

The formulation of NP-proofs restricts the "effective" length of proofs to be polynomial in length of the corresponding assertions (since the running-time of the verification procedure is restricted to be polynomial in the length of the assertion). However, longer proofs may be allowed by padding the assertion with sufficiently many blank symbols. So it seems that NP gives a satisfactory formulation of proof systems (with efficient verification procedures). This is indeed the case if one associates efficient procedures with

[8] NP represents sets of assertions that can be efficiently verified with the help of adequate proofs, whereas P represents sets of assertions that can be efficiently verified from scratch (i.e., without proofs). Thus, "P different from NP" asserts the existence of assertions that are harder to prove than to be convinced of their correctness when presented with a proof. This means that the notion of a proof is meaningful (i.e., that proofs do help when trying to be convinced of the correctness of assertions).

deterministic polynomial-time algorithms. However, we can gain a lot if we are willing to take a somewhat non-traditional step and allow *probabilistic* verification procedures. In particular:

- Randomized and interactive verification procedures, giving rise to *interactive proof systems*, seem much more powerful than their deterministic counterparts (see Section 3.1).
- Such randomized procedures allow the introduction of *zero-knowledge proofs*, which are of great conceptual and practical interest (see Section 3.2).
- NP-proofs can be efficiently transformed into a (redundant) form (called a *probabilistically checkable proof*) that offers a trade-off between the number of bit-locations examined in the NP-proof and the confidence in its validity (see Section 3.3).

In all the abovementioned types of probabilistic proof systems, explicit bounds are imposed on the computational resources 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.*

Clarifications. Like the definition of NP-proof systems, the abovementioned types of probabilistic proof systems refer to proving membership in predetermined sets of strings. That is, the assertions are all of the form "the string x is in a set S", where S is a fixed infinite set and x is a variable input. The definition of an interactive proof system makes explicit reference to a prover, which is only implicit in the definition of an NP-proof system (where the prover is the unmentioned entity providing the proof). We note that, as a first approximation, we are not concerned with the complexity of the prover or the proving task. Our main focus is on the complexity of verification. This is consistent with the intuitive notion of a proof, which refers to the validity of the proof and not to how it was obtained.

3.1 Interactive Proof Systems

In light of the growing acceptability of randomized and distributed 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 "tricky" questions asked by the verifier, to which the prover has to reply "convincingly". The above discussion, as well as the following definition, makes explicit reference to a prover, whereas a prover is only implicit in the traditional definitions of proof systems (e.g., NP-proofs).

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 is probabilistic polynomial-time. 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.

Definition 3.2 (Interactive Proofs – IP [14]): *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 *which executes a computationally unbounded strategy* (denoted P), *satisfying*

- 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 possible strategy P^*, the verifier V rejects with probability at least $\frac{1}{2}$, after interacting with P^* on common input x.*

The class of sets having interactive proof systems is denoted by $\mathcal{IP}$.

Recall that the error probability in the soundness condition can be reduced by successive application of the proof system. To clarify the definition and illustrate the power of the underlying concept, we consider the following story.

> 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.

Note that the proof system underlying this story establishes the dissimilarity of two objects. This idea can be used to provide an interactive proof system for the set of "pairs of non-isomorphic graphs" [12], which informally refer to the dissimilarity of two given objects.[9] Indeed, 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. More generally, it is typically easy to prove the existence of an easily verifiable structure in the given object by merely presenting this structure, but proving the non-existence of such a structure seems hard.

Formally speaking, proving the existence of an easily verifiable structure corresponds to NP-proof systems. The forgoing discussion suggests that interactive proof systems can be used to demonstrate the non-existence of such structures. Specifically, the set of pairs of non-isomorphic graphs is *not* known to have an NP-proof system, and does have an interactive proof system. In general, interactive proof systems can be used to prove the non-existence of any easily verifiable structure; that is, for every $S \in \mathcal{NP}$, the set $\{0,1\}^* \setminus S$ has an interactive proof system (i.e., the class co$\mathcal{NP}$ is contained in $\mathcal{IP}$). We stress that it is widely believed that $\mathrm{co}\mathcal{NP} \stackrel{\mathrm{def}}{=} \{\{0,1\}^* \setminus S : S \in \mathcal{NP}\}$ is *not* contained in $\mathcal{NP}$. For example, the set of systems of quadratic equations that have no integer solutions has an interactive proof system, but is believed not to have an NP-proof system. Furthermore, the class of sets having interactive proof systems coincides with the class $\mathcal{PSPACE}$

[9] A graph $G = (V, E)$ consists of a finite set of vertices V and a finite set of edges E, where each edge is an unordered pair of vertices. 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 $\phi : V_1 \to V_2$ such that $\{u, v\} \in E_1$ if and only if $\{\phi(u), \phi(v)\} \in E_1$.

containing all sets for which membership is decidable by an algorithm that uses a polynomial amount of work-space.

Theorem 3.3 [18, 24]: $\mathcal{IP} = \mathcal{PSPACE}$.

We mention that $\mathcal{NP} \cup \text{co}\mathcal{NP} \subseteq \mathcal{PSPACE}$ and that it is widely believed that $\mathcal{NP}$ contain "little" of $\mathcal{PSPACE}$. Thus, interactive proofs seem to be more powerful than NP-proofs. This conforms with our daily experience by which interaction facilitates the verification of assertions. As we shall argue next, randomness (and the error probability in the soundness condition) play a key role in this phenomenon.

Interactive proof systems extend NP-proof systems in allowing extensive interaction as well as randomization (and ruling based on statistical evidence). As hinted, extensive interaction by itself does not provide any gain (over NP-proof systems). The reason being that the prover can predict the verifier's part of the interaction and thus it suffices to let the prover send the full transcript of the interaction and let the verifier check that the interaction is indeed valid.[10] The moral is that *there is no point to interact with predictable parties that are also computationally weaker.* This moral represents the prover's point of view (with respect to deterministic verifiers). Certainly, from the verifier's point of view it is beneficial to interact with the prover, since it is computationally stronger.

We mention that the power of interactive proof systems remains unchanged under several natural variants. In particular, it turns out that, in this context, *asking clever questions is not more powerful than asking totally random questions.* The reason being that a powerful prover may assist the verifier, which may thus avoid being clever, while the verifier can check (by using only random questions) that the help extended to it is indeed valid. Also, the power of interactive proof systems remains unchanged when allowing two-sided error probability (i.e., allowing bounded error probability also in the completeness condition). Recall that, in contrast, one-sided error probability (i.e., error probability in the soundness condition) is essential to the power of interactive proofs.

3.2 Zero-Knowledge Proof Systems

Standard proofs are believed to yield knowledge and not merely establish the validity of the assertion being proven. Indeed, it is commonly believed

[10] In case the verifier is not deterministic, the transcript sent by the prover may not match the outcome of the verifier coin tosses.

that (good) proofs provide a deeper understanding of the theorem being proved. At the technical level, assuming that NP-proof are useful at all (i.e., assuming that $\mathcal{P} \neq \mathcal{NP}$), an NP-proof of membership in some sets $S \in \mathcal{NP} \setminus \mathcal{P}$ yields something (i.e., the NP-proof itself) that is typically hard to find (even when assuming that the input is in S). For example, an integer solution to a system of quadratic equations constitutes an NP-proof that this system has an integer solution, but it yields information (i.e., the solution) that is infeasible to find (when given an arbitrary system of quadratic equations that has an integer solution). In contrast to such NP-proofs, which seem to yield a lot of knowledge, zero-knowledge proofs yield no knowledge at all; that is, the latter exhibit an extreme contrast between being convincing (of the validity of a statement) and teaching something on top of the validity of the statement.

Loosely speaking, zero-knowledge proofs are interactive proofs that yield nothing beyond the validity of the assertion. These proofs, introduced in [14], are fascinating and extremely useful constructs. Their fascinating nature is due to their seemingly contradictory definition: zero-knowledge proofs are both convincing and yet yield nothing beyond the validity of the assertion being proven. Their applicability in the domain of Cryptography is vast; they are typically used to force malicious parties to behave according to a predetermined protocol. In addition to their direct applicability in Cryptography, zero-knowledge proofs serve as a good bench-mark for the study of various problems regarding cryptographic protocols.

Zero-knowledge is a property of some interactive proof systems, or more accurately of some specified prover strategies. Specifically, it is the property of yielding nothing beyond the validity of the assertion; that is, a verifier obtaining a zero-knowledge 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. Details follow.

The formulation of the zero-knowledge condition refers to two types of probability ensembles, where each ensemble associates a distribution to each valid assertion. The first ensemble represents the output distribution of the verifier after interacting with the specified prover strategy P, where the verifier is not necessarily employing the specified strategy (i.e., V) but rather any efficient strategy. The second ensemble represents the output distribution of some probabilistic polynomial-time algorithm (which 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 3.4 (perfect zero-knowledge, a simplified version[11]): *A prover strategy, P, 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, M^*, such that for every $x \in S$ it holds that $(P, V^*)(x) \equiv M^*(x)$, where $(P, V^*)(x)$ denote the distribution that represents the output of verifier V^* after interacting with the prover P on common input x.*[12]

A somewhat more relaxed interpretation of similarity, leading to almost-perfect 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, is that similarity means computational indistinguishability (i.e., failure of any efficient procedure to tell the two ensembles apart). The actual definition is obtained from Definition 2.1, by considering ensembles indexed by strings and providing the distinguisher with the relevant index. That is, *the probability ensembles, $\{Y_x\}_{x\in S}$ and $\{Z_x\}_{x\in S}$, are* indistinguishable *by an algorithm A if*

$$d_A(n) \stackrel{\rm def}{=} \max_{x\in S\cap\{0,1\}^n} \{|prob(A(x, Y_x)=1) - \mathsf{Pr}(A(x, Z_x)=1)|\}$$

is a negligible function.[13] *The ensembles $\{Y_x\}_{x\in S}$ and $\{Z_x\}_{x\in S}$ are* computationally indistinguishable *if they are indistinguishable by every probabilistic polynomial-time algorithm.*

The foregoing discussion refers to simplified versions of the actual definitions. Specifically, in order to guarantee that zero-knowledge is preserved under sequential composition it is necessary to slightly augment the definitions. For details see [8, Sec. 4.3.3-4.3.4].

[11] The actual definition allows for a rare event (which occurs with negligible probability) in which M^* halts with no output, and the output of M^* is considered condition on this event not occuring.

[12] As usual, $M^*(x)$ denotes a distribution representing the output of algorithm M^* on input x.

[13] If $S \cap \{0,1\}^n = \emptyset$ then we define $d_A(n) = 0$.

The Power of Zero-Knowledge. We consider the set of 3-colorable graphs, where a graph[14] $G=(V,E)$ is said to be *3-colorable* if there exists a function $\pi:V\to\{1,2,3\}$ (called a *3-coloring*) such that $\pi(v)\neq\pi(u)$ for every $\{u,v\}\in E$. It is easy to prove that a given graph G is 3-colorable by just presenting a 3-coloring of G, but this NP-proof is not a zero-knowledge proof (unless $\mathcal{P}=\mathcal{NP}$). In fact, assuming $\mathcal{P}\neq\mathcal{NP}$, graph 3-colorability has no zero-knowledge NP-proofs, but as we shall see it has zero-knowledge interactive proofs. We first describe these proof systems using (abstract) "boxes" in which information can be hidden and later revealed. Such "boxes" can be implemented using one-way functions.

Construction 3.5 (Zero-knowledge proof of 3-colorability [12]): *On common input, $G=(V,E)$, The following steps are repeated $|V|\cdot|E|$ times.*

- 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 $\phi(v)\stackrel{\rm def}{=}\pi(\psi(v))$, for each $v\in V$. Hence, the prover forms a random relabeling of the 3-coloring ψ. The prover sends the verifier a sequence of $|V|$ locked and non-transparent boxes such that the $v^{\rm th}$ 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. Intuitively, the verifier asks to inspect the colors of vertices u and v.*
- Prover's second step: *The prover sends to the verifier the keys to boxes u and v.*
- Verifier's second step: *The verifier opens boxes u and v, and checks whether or not they contain two different elements in $\{1,2,3\}$.*

The verifier accepts if and only if all checks turn out positive.

The foregoing verifier strategy is easily implemented in probabilistic polynomial-time. The same holds with respect to the prover's strategy, provided it is given a 3-coloring of G as auxiliary input. Clearly, if the input graph is 3-colorable then the prover can cause the verifier to accept with probability 1. On the other hand, if the input graph is not 3-colorable then any contents put in the boxes must be invalid on at least one edge, and consequently each time the foregoing steps are repeated the verifier rejects with probability at

[14] See Footnote 9.

least $\frac{1}{|E|}$. Repeating these steps $t \cdot |E|$ times has the effect of reducing the soundness error probability to

$$\left(1 - \frac{1}{|E|}\right)^{t \cdot |E|} \approx e^{-t}.$$

The zero-knowledge property follows easily, in this abstract setting, because one can simulate the real interaction by placing a random pair of different colors in the boxes indicated by the verifier. This indeed demonstrates that the verifier learns nothing from the interaction (since it expects to see a random pair of different colors and indeed this is what it sees). We stress that this simple argument is not possible in the digital implementation because the boxes are not totally unaffected by their contents (but are rather affected, yet in an indistinguishable manner).

As stated, in order to obtain a real interactive proof, the (abstract or physical) "boxes" need to be implemented digitally. This can be done using an adequately defined "commitment scheme" (see [8, Sec. 4.4.1]). Loosely speaking, such a scheme is a two phase game between a sender and a receiver so 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. 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. Such commitment schemes can be implemented assuming the existence of one-way functions. Thus, the existence of one-way functions implies a zero-knowledge proofs for 3-colorability. In fact, one gets zero-knowledge proofs for any NP-set.

Theorem 3.6 [12]: *Assuming the existence of one-way functions, any NP-proof can be efficiently transformed into a zero-knowledge interactive proof. That is, the prover strategy in the zero-knowledge interactive proof can be implemented in probabilistic polynomial-time provided that it is given an adequate NP-proof as auxiliary input.*

Theorem 3.6 has a dramatic effect on the design of cryptographic protocols (cf., [8, 9]). In a different vein and for the sake of elegance, we mention that, using further ideas and under the same assumption, any set having an interactive proof system also has a zero-knowledge interactive proof system.

The Role of Randomness. Again, randomness is essential to all the aforementioned results. Namely, zero-knowledge proof systems in which either the verifier or the prover is deterministic exist only for sets in $\mathcal{BPP}$, where $\mathcal{BPP}$ is the class of sets for which membership is decidable by some probabilistic polynomial-time algorithm. Note that such sets have trivial zero-knowledge proofs in which the prover sends nothing and the verifier just test the validity of the assertion by itself. Thus, randomness is essential to the usefulness of zero-knowledge proofs.

3.3 Probabilistically Checkable Proof Systems

When viewed in terms of an interactive proof system, the probabilistically checkable proof setting consists of a prover that is memoryless (and responds to each verifier message as if it were the first such message). However, it is more appealing to view probabilistically checkable proof systems 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. In fact, we focus on the latter probabilistic procedure, which is given direct access to the individual bits of the alleged proof (and need not scan it bit-by-bit). Thus, the alleged proof is a string, as in the case of a traditional proof system, but 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.

The main complexity measure associated with probabilistically checkable proof (PCP) systems is indeed their query complexity (i.e., the number of queries bits accessed in the alleged proof). 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 non-approximability results), and thus we specify this parameter rather than the proof length.

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 Item (1) of Definition 3.7. Yet, the complexity measures introduced in Item (2) are of key importance for the subsequent discussions.

Definition 3.7 (Probabilistically Checkable Proofs – PCP):

1. *A* probabilistically checkable proof system (PCP) for a set S *is a probabilistic polynomial-time oracle machine* (called verifier), *denoted* V*, satisfying*

   - Completeness*: For every* $x \in S$ *there exists an oracle* π_x *so that* V*, on input* x *and access to* π_x*, always accepts* x*.*
   - Soundness*: For every* $x \notin S$ *and every oracle* π*, machine* V*, on input* x *and access to* π*, rejects* x *with probability at least* $\frac{1}{2}$*.*

2. *Let* r *and* q *be integer functions. The complexity class* $\mathcal{PCP}(r(\cdot), q(\cdot))$ *consists of sets having a probabilistically checkable proof system in which the verifier, on any input of length* n*, makes at most* $r(n)$ *coin tosses and at most* $q(n)$ *oracle queries, where each query is answered by a single bit. For sets of integer functions,* R *and* Q*, we let* $\mathcal{PCP}(R, Q)$ *equal* $\cup_{r\in R,q\in Q}\mathcal{PCP}(r(\cdot), q(\cdot))$*.*

We stress that the oracle π_x in a PCP system constitutes a proof in the standard mathematical sense. Yet, this oracle has the extra 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).

Letting `poly` denote the set of all polynomials, one may verify that $\mathcal{PCP}(0, \texttt{poly}) = \mathcal{NP}$. Letting `log` denote the set of all logarithmic functions (i.e., $\ell \in \texttt{log}$ if there exists a constant b such that $\ell(n) \le \log_b n$ for all sufficiently large n), one may also verify that $\mathcal{PCP}(\texttt{log}, \texttt{poly}) \subseteq \mathcal{NP}$ (because the relevant oracles are of polynomial length). It follows that, for every constant c, it holds that $\mathcal{PCP}(\texttt{log}, c) \subseteq \mathcal{NP}$. This upper bound turned out to be tight, but proving this is much more difficult (to say the least). The following result is a culmination of a sequence of great works (see [7, Sec. 2.6.2] for a detailed account).

Theorem 3.8 [2, 1]: *There exists a constant* c *such that* $\mathcal{NP} \subseteq \mathcal{PCP}(\texttt{log}, c)$*.*

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 the complexity class $\mathcal{NP}$. (Essentially, this constant is three.) Furthermore, NP-proofs can be efficiently transformed into NP-proofs that offer a trade-off between the portion of the proof being read and the confidence it offers. Specifically, if the verifier is willing to tolerate an error probability of ϵ then it suffices to let it examine $c \cdot \log_2(1/\epsilon)$ bits of the (transformed) NP-proof.[15] These bit locations need to be selected at random. We mention that the length of the redundant NP-proofs that provide the aforementioned trade-off can be made almost linear in the length of the standard NP-proofs.

PCP and the study of approximation. Following [5] and [1], the characterization of $\mathcal{NP}$ in terms of probabilistically checkable proofs has played a central role in developments concerning the study of approximation problems. For details, see [16, Chap. 10]. We merely mention that Theorem 3.8 implies that, assuming $\mathcal{P} \neq \mathcal{NP}$, there exists a constant $\delta < 1$ such that *given a system of quadratic equations it is infeasible to distinguish the case in which the system has an integer solution from the case that any assignment of integers satisfies at most a δ fraction of the equations.*

The Role of Randomness. No trade-off between the number of bits examined and the confidence is possible if one requires the verifier to be deterministic. In particular, $\mathcal{PCP}(0, \texttt{log}) = \mathcal{P}$.

3.4 Suggestions for further reading

More detailed overviews of the three types of probabilistically proof systems can be found in [7, Chap. 2]. A detailed textbook treatment of zero-knowledge is provided in [8, Chap. 4].

4 Cryptography

In this section, we focus on the role of randomness in Cryptography. As stated at the beginning of the introduction, the very notion of a secret,

[15] In fact, c can be made arbitrarily close to one, when ϵ is small enough.

which is central to Cryptography, refers to randomness in the sense of unpredictability (i.e., unpredictability of the secret by other parties). Furthermore, the use of randomized algorithms and/or strategies is essential for achieving almost any security concern. We start with the concrete example of providing secret and authenticated communication, and end with a wider perspective.

4.1 Secret and authenticated communication

The problem of providing *secret communication over insecure media* is the traditional and most basic problem of Cryptography. The setting of this problem consists of two parties communicating through a channel that is possibly tapped by an adversary. The parties wish to exchange information with each other, but keep the "wire-tapper" as ignorant as possible regarding the contents of this information. The canonical solution to the above problem is obtained by the use of encryption schemes.

Loosely speaking, an encryption scheme is a protocol allowing these parties to communicate *secretly* with each other. Typically, the encryption scheme consists of a pair of algorithms. One algorithm, called encryption, is applied by the sender (i.e., the party sending a message), while the other algorithm, called decryption, is applied by the receiver. Hence, in order to send a message, the sender first applies the encryption algorithm to the message, and sends the result, called the ciphertext, over the channel. Upon receiving a ciphertext, the other party (i.e., the receiver) applies the decryption algorithm to it, and retrieves the original message (called the plaintext).

In order for the foregoing scheme to provide secret communication, the communicating parties (at least the receiver) must know something that is not known to the wire-tapper. (Otherwise, the wire-tapper can decrypt the ciphertext exactly as done by the receiver.) This extra knowledge may take the form of the decryption algorithm itself, or some parameters and/or auxiliary inputs used by the decryption algorithm. We call this extra knowledge the decryption-key. Note that, without loss of generality, we may assume that the decryption algorithm is known to the wire-tapper, and that the decryption algorithm operates on two inputs: a ciphertext and a decryption-key. (The encryption algorithm also takes two inputs: a corresponding encryption-key and a plaintext.) We stress that the existence of a decryption-key, not known to the wire-tapper, is merely a necessary condition for secret communication.

The point we wish to make is that the decryption-key must be generated by a randomized algorithm. Suppose, in contrary, that the decryption-key is a predetermined function of publicly available data (i.e., the key is generated by employing an efficient deterministic algorithm to this data). Then, the wire-tapper can just obtain the key in exactly the same manner (i.e., invoking the same algorithm on the said data). We stress that saying that the wire-tapper does not know which algorithm to employ or does not have the data on which the algorithm is employed just shifts the problem elsewhere; that is, the question remains as to *how do the legitimate parties select this algorithm and/or the data to which it is applied*? Again, deterministically selecting these objects based on publicly available data will not do. At some point, *the legitimate parties must obtain some object that is unpredictable by the wire-tapper*, and such unpredictability refers to randomness (or pseudorandomness).

However, the role of randomness in allowing for secret communication is not confined to the generation of secret keys. To see why this is the case, we need to understand what is "secrecy" (i.e., to properly define what is meant by this intuitive term). Loosely speaking, we say that an encryption scheme is secure if it is *infeasible for the wire-tapper to obtain from the ciphertexts any additional information about the corresponding plaintexts*. In other words, whatever can be efficiently computed based on the ciphertexts can be efficiently computed from scratch (or rather from the a priori known data). Now, assuming that the encryption algorithm is deterministic, encrypting the same plaintext twice (using the same encryption-key) results in two identical ciphertexts, which are easily distinguishable from any pair of different ciphertexts resulting from the encryption of two different plaintexts. This problem does not arise when employing a randomized encryption algorithm (as presented next).

As hinted, an encryption scheme must specify also a method for selecting keys. In the following encryption scheme, a uniformly chosen n-bit key, s, is used for specifying a pseudorandom function f_s (as in Definition 2.7). A plaintext $x \in \{0,1\}^n$ is encrypted (using the key s) by uniformly selecting $r \in \{0,1\}^n$ and producing the ciphertext $(r, f_s(r) \oplus x)$, where $\alpha \oplus \beta$ denotes the bit-by-bit exclusive-or of the strings α and β. A ciphertext (r, y) is decrypted (using the key s) by computing $f_s(r) \oplus y$. The security of this scheme follows from the security of an imaginary (ideal) scheme in which f_s is replaced by a totally random function $F : \{0,1\}^n \to \{0,1\}^n$.

Public-key encryption schemes. The foregoing description corresponds to the so called model of a *private-key encryption scheme*, and requires the communicating parties to agree beforehand on a corresponding pair of encryption/decryption keys. This need is removed in *public-key encryption schemes*, envisioned by Diffie and Hellman (and materialized by the RSA scheme of Rivest, Shamir, and Adleman). In a public-key encryption scheme, the encryption-key can be publicized without harming the security of the plaintexts encrypted using it, allowing anybody to send encrypted messages to Party X by using the encryption-key publicized by Party X. But in such a case, the need for randomized encryption is even more clear. Indeed, if a deterministic encryption algorithm is employed and the wire-tapper knows the encryption-key, then it can identity of the plaintext in the case that the number of possibilities is small. In contrast, using a randomized encryption algorithm, the encryption of plaintext `yes` under a known encryption-key may be computationally indistinguishable from the encryption of the plaintext `no` under the say encryption-key. For further discussion of the security and construction of encryption schemes, the interested reader is referred to [9, Chap. 5].

Authenticated communication. Message authentication is a task related to the setting considered for private-key encryption schemes. Again, there are two designated parties that wish to communicate over an insecure channel. This time, we consider an active adversary that is monitoring the channel and may alter the messages sent on it. The parties communicating through this insecure channel wish to authenticate the messages they send such that their counterpart can tell an original message (sent by the sender) from a modified one (i.e., modified by the adversary). Loosely speaking, a scheme for message authentication should satisfy the following:

- each of the communicating parties can *efficiently produce an authentication tag* to any message of its choice;
- each of the communicating parties can *efficiently verify* whether a given string is an authentication tag of a given message; but
- *it is infeasible for an external adversary* (i.e., a party other than the communicating parties) *to produce authentication tags* to messages not sent by the communicating parties.

Again, such a scheme consists of a randomized algorithm for selecting keys as well as algorithms for tagging messages and verifying the validity of tags.

In the following message authentication scheme, a uniformly chosen n-bit key, s, is used for specifying a pseudorandom function (as in Definition 2.7). Using the key s, a plaintext $x \in \{0,1\}^n$ is authenticated by the tag $f_s(x)$, and verification of (x, y) with respect to the key s amounts to checking whether y equals $f_s(x)$. For further discussion of message authentication schemes and the related notion of signature schemes, the interested reader is referred to [9, Chap. 6].

4.2 A wider perspective

Modern Cryptography is concerned with the construction of information systems that are robust against malicious attempts to make these systems deviate from their prescribed functionality. The prescribed functionality may be the private and authenticated communication of information through the Internet, the holding of incoercible and secret electronic voting, or conducting any "fault-resilient" multi-party computation. Indeed, the scope of modern Cryptography is very broad, and it stands in contrast to "classical" Cryptography (which has focused on the single problem of enabling secret communication over insecure communication media).

The design of cryptographic systems is a very difficult task. One cannot rely on intuitions regarding the "typical" state of the environment in which the system operates. For sure, the adversary attacking the system will try to manipulate the environment into "untypical" states. Nor can one be content with counter-measures designed to withstand specific attacks, since the adversary (which acts after the design of the system is completed) will try to attack the schemes in ways that are different from the ones the designer had envisioned. The validity of the above assertions seems self-evident, still some people hope that in practice ignoring these tautologies will not result in actual damage. Experience shows that these hopes rarely come true; cryptographic schemes based on make-believe are broken, typically sooner than later.

In view of the foregoing, we believe that it makes little sense to make assumptions regarding the specific *strategy* that the adversary may use. The only assumptions that can be justified refer to the computational *abilities* of the adversary. Furthermore, the design of cryptographic systems has to be based on *firm foundations*; whereas ad-hoc approaches and heuristics are a very dangerous way to go. A heuristic may make sense when the designer has a very good idea regarding the environment in which a scheme is to operate, yet a cryptographic scheme has to operate in a maliciously selected environment which typically transcends the designer's view.

The foundations of Cryptography are the paradigms, approaches and techniques used to conceptualize, define and provide solutions to natural "security concerns". For a presentation of these foundations, the interested reader is referred to [8, 9]. Here we merely note that randomness plays a central role in each definition and technique presented there. In almost every case, the inputs of the legitimate parties are assumed to be unpredictable by the adversary, and the task is performing some manipulation (of the inputs) while preserving or creating some unpredictability. In all cases, this is obtained by using randomized algorithms.

5 Sub-linear time algorithms

For starters, let us consider a well-known example in which fast approximations are possible and useful. Suppose that some cost function is defined over a huge data-set, and that one wants to approximate the average cost of an element in the set. To be more specific, let $\mu : S \to [0,1]$ be a cost function, and suppose we want to estimate $\overline{\mu} \stackrel{\rm def}{=} \frac{1}{|S|} \sum_{e \in S} \mu(e)$. Then, for some constant c, uniformly (and independently) selecting $m \stackrel{\rm def}{=} c \cdot \varepsilon^{-2} \log_2(1/\delta)$ sample points, $s_1, ..., s_m$, in S we obtain with probability at least $1 - \delta$ an estimate of $\overline{\mu}$ within $\pm\varepsilon$:

$$\mathsf{Pr}_{s_1,...,s_m \in S} \left[\left| \frac{1}{m} \sum_{i=1}^{m} \mu(s_i) \; - \; \overline{\mu} \right| \; > \; \varepsilon \right] \;\; < \;\; \delta \, .$$

We stress the fact that the number of samples *only depends on the desired level of approximation* (and is independent of the size of S). In this section we discuss analogous phenomena that occur with respect to objectives that are beyond gathering statistics of individual values. We focus on more complex features of a data-set; specifically, relations among pairs of elements rather than values of single elements. Such binary relations are captured by graphs (as defined in Footnote 9); that is, a symmetric binary relation $R \subseteq S \times S$ is represented by a graph $G = (S, R)$, where the elements of S are called vertices and the elements of R are called edges. Each edge consists of a pair of vertices, called its end-points.

One natural computational question regarding graphs is whether or not they are bi-partite; that is, whether there exists a partition of S into two subsets S_1 and S_2 such that each edge has one end-point in S_1 and the other endpoint in S_2. For example, the graph consisting of a cycle of four vertices is bi-partite, whereas a triangle is not bi-partite. We mention that there exists an efficient algorithm that given a graph G determines whether or not G is bi-partite. Needless to say, this algorithm must inspect all edges of G, whereas we seek sub-linear time algorithms (i.e., algorithms operating in time smaller than the size of the input). In particular, sub-linear time algorithms cannot afford reading the entire input graph. Instead, these algorithm can inspect portions of the input graph by querying for the existence of specific edges (i.e., query whether there is an edge between a specific pair of vertices). It turns out that, by making a number of queries that is independent of the size of the graph, one may obtain meaningful information regarding its "distance" to being bi-partite. Specifically:

Theorem 5.1 [11]: *There exists a randomized algorithm that, on input a parameter ε and access to a graph $G = (S, R)$, makes* $\text{poly}(1/\varepsilon)$ *queries to G and satisfies the following two conditions:*

1. *If G is bi-partite then the algorithm accepts with probability 1.*
2. *If any partition of S into two subsets S_1 and S_2 has at least $\varepsilon|S|^2$ edges with both end-points in the same S_i then the algorithm rejects with probability at least 99%.*

The algorithm underlying Theorem 5.1 uniformly selects $m = \text{poly}(1/\varepsilon)$ vertices, and checks whether the induced graph is bi-partite; that is, for a sample of vertices $v_1, ..., v_m$, it checks whether there exists a partition of $\{v_1, ..., v_m\}$ into two subsets V_1 and V_2 such that for every $i \in \{1, 2\}$ and every $u, v \in V_i$ it holds that $(u, v) \notin R$.

We stress that the said algorithm does not solve the question of whether or not the graph is bi-partite, but rather a relaxed (or approximated) version of this question in which one needs to *distinguish graphs that are bi-partite from graphs that a very far from being bi-partite.* This phenomenon is analogous to the case of approximating the average value of $\mu : S \to [0, 1]$. Also, as in the case of approximating the average value of $\mu : S \to [0, 1]$, it is essential that the approximation algorithm be randomized. A similar phenomena occurs with respect to several other natural properties of graphs, but is not generic. That is, there exist graph properties for which even inspecting a constant fraction of the graph does not allow for an approximate decision regarding satisfiability of the property. For details, the interested reader is directed to [6, 21].

We note that the notion of approximation underlying Theorem 5.1 refers to disregarding $\varepsilon|S|^2$ edges, where $|S|^2$ is the maximum possible number of edges over S. This notion of approximation is appealing in the case that R is dense (i.e., contains a constant fraction of all possible edges). Going to the other extreme, we may consider the case that R contains only a linear in S number of edge, or even the case that each vertex participates only in a constant number of edges. In this case, we may want to distinguish the case that the graph is bi-partite from the case that any partition of S into two subsets S_1 and S_2 has at least $\varepsilon|S|$ edges with both end-points in the same S_i. It turns out that this problem can be solved by an algorithm that makes $\mathrm{poly}((\log|S|)/\varepsilon)\cdot\sqrt{|S|}$ queries (to an adequate data structure), and that these many queries are essentially necessary. We note that this sub-linear time algorithm operates by inspecting a graph induced by $\mathrm{poly}((\log|S|)/\varepsilon)\cdot\sqrt{|S|}$ vertices that are selected by taking many (relatively short) random walks from few randomly selected starting vertices. For details, the interested reader is directed to [21, Sec. 3].

The aforementioned type of approximation is known by the name *property testing*, and was initiated and developed in [22, 11]. One archetypical problem, which played a central role in the construction of PCP systems (see Section 3.3), is distinguishing low-degree polynomials from functions that are far from any such polynomial. Specifically, let F be a finite field and m, d be integers. Given access to a function $f : F^m \to F$, we wish to make few queries and distinguish the case that f is am m-variate polynomial of total degree d from the case it disagrees with any such polynomial on at least 1% of the domain. It turns out that making $\mathrm{poly}(d)$ random (but dependent) queries to f suffices for making a decision that is correct with high probability.

Acknowledgments

I am grateful to Tamas Rudas for his comments on early versions of this essay.

References

[1] S. Arora, C. Lund, R. Motwani, M. Sudan and M. Szegedy. Proof Verification and Intractability of Approximation Problems. *Journal of the ACM*, Vol. 45, pages 501–555, 1998.

[2] S. Arora and S. Safra. Probabilistic Checkable Proofs: A New Characterization of NP. *Journal of the ACM*, Vol. 45, pages 70–122, 1998.

[3] M. Blum and S. Micali. How to Generate Cryptographically Strong Sequences of Pseudo-Random Bits. *SIAM Journal on Computing*, Vol. 13, pages 850–864, 1984.

[4] T.M. Cover and G.A. Thomas. *Elements of Information Theory*. John Wiley & Sons, Inc., New-York, 1991.

[5] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy. Approximating Clique is almost NP-complete. *Journal of the ACM*, Vol. 43, pages 268–292, 1996.

[6] E. Fischer. The art of uninformed decisions: A primer to property testing. *Bulletin of the European Association for Theoretical Computer Science*, Vol. 75, pages 97–126, 2001.

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

[8] O. Goldreich. *Foundation of Cryptography – Basic Tools*. Cambridge University Press, 2001.

[9] O. Goldreich. *Foundation of Cryptography: Basic Applications*. Cambridge University Press, 2004.

[10] O. Goldreich, S. Goldwasser, and S. Micali. How to Construct Random Functions. *Journal of the ACM*, Vol. 33, No. 4, pages 792–807, 1986.

[11] O. Goldreich, S. Goldwasser, and D. Ron. Property testing and its connection to learning and approximation. *Journal of the ACM*, pages 653–750, July 1998.

[12] O. Goldreich, S. Micali and A. Wigderson. Proofs that Yield Nothing but their Validity or All Languages in NP Have Zero-Knowledge Proof Systems. *Journal of the ACM*, Vol. 38, No. 3, pages 691–729, 1991.

[13] S. Goldwasser and S. Micali. Probabilistic Encryption. *Journal of Computer and System Science*, Vol. 28, No. 2, pages 270–299, 1984.

[14] S. Goldwasser, S. Micali and C. Rackoff. The Knowledge Complexity of Interactive Proof Systems. *SIAM Journal on Computing*, Vol. 18, pages 186–208, 1989.

[15] J. Håstad, R. Impagliazzo, L.A. Levin and M. Luby. A Pseudorandom Generator from any One-way Function. *SIAM Journal on Computing*, Volume 28, Number 4, pages 1364–1396, 1999.

[16] D. Hochbaum (ed.). *Approximation Algorithms for NP-hard Problems.* PWS, 1996.

[17] M. Li and P. Vitanyi. *An Introduction to Kolmogorov Complexity and its Applications.* Springer Verlag, August 1993.

[18] C. Lund, L. Fortnow, H. Karloff, and N. Nisan. Algebraic Methods for Interactive Proof Systems. *Journal of the ACM*, Vol. 39, No. 4, pages 859–868, 1992.

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

[20] C.H. Papadimitriou. *Computational Complexity.* Addison Wesley, 1994.

[21] D. Ron. Property testing. In *Handbook on Randomization, Volume II*, pages 597–649, 2001. (Eds: S. Rajasekaran, P.M. Pardalos, J.H. Reif and J.D.P. Rolim.)

[22] R. Rubinfeld and M. Sudan. Robust characterization of polynomials with applications to program testing. *SIAM Journal on Computing*, Vol. 25 (2), pages 252–271, 1996.

[23] A. Shamir. IP = PSPACE. *Journal of the ACM*, Vol. 39, No. 4, pages 869–877, 1992.

[24] M. Sipser. *Introduction to the Theory of Computation.* PWS Publishing Company, 1997.

[25] A.C. Yao. Theory and Application of Trapdoor Functions. In *23rd IEEE Symposium on Foundations of Computer Science*, pages 80–91, 1982.