Foundations and Trends® in
Theoretical Computer Science
Vol. 2, No 1 (2006) 1–106

DOI: 10.1561/0400000004

now
the essence of knowledge

Average-Case Complexity

Andrej Bogdanov[1] and Luca Trevisan[2]

[1] *DIMACS – Rutgers University, adib@dimacs.rutgers.edu*
[2] *UC Berkeley, luca@eecs.berkeley.edu*

Abstract

We survey the average-case complexity of problems in NP.

We discuss various notions of good-on-average algorithms, and present completeness results due to Impagliazzo and Levin. Such completeness results establish the fact that if a certain specific (but somewhat artificial) NP problem is easy-on-average with respect to the uniform distribution, then all problems in NP are easy-on-average with respect to all samplable distributions. Applying the theory to natural distributional problems remain an outstanding open question. We review some natural distributional problems whose average-case complexity is of particular interest and that do not yet fit into this theory.

A major open question is whether the existence of hard-on-average problems in NP can be based on the $\mathrm{P} \neq \mathrm{NP}$ assumption or on related worst-case assumptions. We review negative results showing that certain proof techniques cannot prove such a result. While the relation between worst-case and average-case complexity for general NP problems remains open, there has been progress in understanding the relation between different "degrees" of average-case complexity. We discuss some of these "hardness amplification" results.

1

Introduction

The study of the average-case complexity of intractable problems began in the 1970s motivated by two distinct applications: the development of the foundations of cryptography and the search for methods to "cope" with the intractability of NP-hard problems.

All definitions of security for cryptographic problems require that any efficient algorithm that tries to "break" the protocol "succeeds" only with a very small probability. The formalizations of *breaking* and *succeeding* depend on the specific application, but it has been known since the 1980s that there is a unifying concept: no cryptographic task (e.g., electronic signature or data encryption) is possible unless *one-way functions* exist.[1] Informally, a one-way function is an efficiently computable function $f : \{0,1\}^* \to \{0,1\}^*$ that maps $\{0,1\}^n$ to $\{0,1\}^n$ and such that, if we are given $f(x)$ for a random $x \in \{0,1\}^n$, it is intractable (in time polynomial in n) to find a pre-image x' such that $f(x') = f(x)$. In particular, the existence of one-way functions implies that there is a search problem in NP (given $y \in \{0,1\}^n$, find $x \in \{0,1\}^n$ such that $f(x) = y$) that is intractable to solve on random inputs sampled from

[1] The realizability of many cryptographic tasks, in fact, is *equivalent* to the assumption that one-way functions exist.

a simple distribution (the distribution $f(x)$, where x is chosen randomly from $\{0,1\}^n$). The fact that all of cryptography is predicated on the existence of average-case intractable problems in NP is a main motivation for the study of the theory we describe in this study.

In particular, a long-standing open question is whether it is possible to *base the existence of one-way functions on the* $\mathrm{P} \neq \mathrm{NP}$ *assumption*, or related ones (such as NP-complete problems not allowing polynomial size circuits).

The second motivation for the study of the average-case complexity of problems in NP comes from the analysis of heuristic algorithms. Unless $\mathrm{P} = \mathrm{NP}$, we cannot hope for efficient algorithms that solve NP-complete problems exactly on all inputs. We may hope, however, for algorithms that are "typically efficient" on inputs sampled from distributions that occur in practice. In order to understand the limitations of such an approach, it would be desirable to have an "average-case analog" of the theory of NP-completeness. Such a theory would enable us to prove that for certain problems, with respect to certain distributions, it is impossible to have algorithms that perform well on "typical" inputs, unless an entire class of presumably intractable problems can be efficiently solved.

The basic foundations of such a theory have been laid out. Surprisingly, subtle difficulties arise even when just developing the analogs of trivial elements of the theory of NP-completeness, such as the definitions of *computational problem, efficient algorithm, reduction*, and *completeness*, and the equivalent complexity of *decision versus search* for NP-complete problems. In this study we will discuss these difficulties and show how they were resolved. We will see a number of results, insights, and proof techniques the usefulness of which goes beyond the study of average-case complexity.

The right techniques to apply such a theory to natural problems and distributions have not been discovered yet. From this point of view, the current state of the theory of average-case complexity in NP is similar to the state of the theory of inapproximability of NP optimization problems before the PCP Theorem.

Finding ways of *applying this theory to natural problems* is another outstanding open question in this area.

1.1 Roadmap

In this section we give an overview of the content of this survey.

1.1.1 Definitions of tractability

The first difficulty in developing a theory of average-case intractability is to come up with a formal definition of what it means for a problem to be "intractable on average" or, equivalently, what it means to be "average-case tractable." A natural definition would be to consider an algorithm efficient-on-average if it runs in *expected polynomial time*. Such a definition has various shortcomings (related to the fact that it is too restrictive). For example, if an algorithm A runs in time $t(x)$, and its simulation B (in a different model of computation) runs in time $t^2(x)$, it is natural that we would like our definition to be such that A is efficient-on-average if and only if B is. Suppose, however, that our inputs come from the uniform distribution, and that A runs in time n^2 on all inputs of length n, except on one input on which A takes time 2^n. Then the expected running time of A is polynomial but the expected running time of B is exponential. Looking at the *median* running time of an algorithm gives us a more robust measure of complexity, but still a very unsatisfactory one: if an algorithm runs in polynomial time on 70% of the inputs, and in exponential time on 30% of the inputs, it seems absurd to consider it an efficient-on-average algorithm. The right way to capture the notion of "efficient on typical instances" should be that it is fine for an algorithm to take a large amount of time on certain inputs, provided that such inputs do not occur with high probability: that is, inputs requiring larger and larger running times should have proportionally smaller and smaller probability. This is the idea of Levin's definition of average-case complexity. In (an equivalent formulation of) Levin's definition [53], an algorithm is polynomial-time-on-average if there is a constant $c > 0$ such that the probability, over inputs of length n, that the algorithm takes more than time T is at most $\text{poly}(n)/T^c$. As is usual with complexity theory, various choices can be made in the definition: we may look at deterministic algorithms, randomized algorithms, or non-uniform families of circuits. An additional choice is whether we require our algorithm to always be

correct, but possibly run in superpolynomial time on some inputs, versus requiring the algorithm to always run in polynomial time, but to give an incorrect answer to some inputs. This will lead to several possible definitions, each meaningful in some applications. (See Chapter 2.) The important thing will be that almost all the results we discuss in this study are based on reductions that preserve tractability under all of these definitions. Hence, the treatment of completeness, reductions, families of distributions, and decision versus search is independent of the specific notion of tractability that one is interested in.

1.1.2 Reductions between distributional problems

Let L be a decision problem and $\mathcal{D}$ be a distribution over inputs[2]; we call the pair $(L, \mathcal{D})$ a *distributional problem*. All the definitions of average-case tractability have a characteristic in common: an algorithm A is efficient for $(L, \mathcal{D})$ if a certain set of "bad" inputs has low probability under $\mathcal{D}$. (The bad inputs could be the ones where the algorithm A takes a very long time, or those on which A outputs an incorrect answer.) This motivates the following definition of reduction [53]: we say that $(L, \mathcal{D})$ reduces to $(L', \mathcal{D}')$ if there is a polynomial time computable function f such that $x \in L$ if and only if $f(x) \in L'$ and, in addition, for every input y, the probability of generating y by picking x at random according to $\mathcal{D}$ and then computing $f(x)$ is at most $\text{poly}(|x|)$ larger than the probability of sampling y at random from $\mathcal{D}'$.[3] The motivation for this definition is the following. Suppose that A' is a good algorithm for $(L', \mathcal{D}')$, so that the set B' of inputs that are bad for A' has a small probability according to $\mathcal{D}'$. Consider the following algorithm for $(L, \mathcal{D})$: on input x, output $A'(f(x))$. Now, the bad inputs for this algorithm are the inputs x such that $f(x) \in B'$. The probability of sampling such an x, according to $\mathcal{D}$, however, is upper bounded by $\text{poly}(|x|)$ times the probability of sampling an element of B' according to $\mathcal{D}'$, which we had assumed to be small. Hence, we have a good algorithm for $(L, \mathcal{D})$, and the definition of reduction preserves average-case tractability. Note that, in this argument, we used nothing about the

[2] Additional difficulties arise in defining how to specify $\mathcal{D}$.
[3] When the second condition holds, we say that $\mathcal{D}'$ *dominates* $\mathcal{D}$.

definition of tractability except the notion of "bad" input. (See also Chapter 3.)

1.1.3 A completeness result

Having given the definition of computational problem and of reduction, we will present a *completeness result* [53]. We consider the *bounded halting* problem BH, where on input $(M, x, 1^t)$ we have to determine whether the non-deterministic Turing machine M accepts input x within t steps. This problem is readily seen to be NP-complete. We show that for every distributional problem $(L, \mathcal{D})$, where L is in NP and $\mathcal{D}$ is a *polynomial-time computable* distribution there is a reduction from $(L, \mathcal{D})$ to $(\mathrm{BH}, \mathcal{U}^{\mathrm{BH}})$, where $\mathcal{U}^{\mathrm{BH}}$ is a reasonable formalization of the notion of a "uniformly chosen" random input for BH. Informally, the reduction maps an input x into the triple $(M', C(x), 1^t)$, where C is a (carefully chosen) injective polynomial-time computable encoding function; M' is a non-deterministic machine that first recovers x from $C(x)$ and then simulates the non-deterministic polynomial time Turing machine that decides whether $x \in L$ (recall that L is in NP); and t is a polynomial upper bound to the running time of M'. The main claim in the analysis of the reduction is that, for x selected from $\mathcal{D}$, $C(x)$ is "approximately" uniformly distributed. Technically, we show that the distribution of $C(x)$ is dominated by the uniform distribution. This will follow from a choice of C as an information-theoretically optimal compression scheme.

The completeness result implies that if $(\mathrm{BH}, \mathcal{U}^{\mathrm{BH}})$ has a good-on-average algorithm (according to one of the possible definitions), then all problems $(L, \mathcal{D})$, where L is in NP and $\mathcal{D}$ is polynomial-time computable, also have good-on-average algorithms.

The proof uses the fact that all *polynomial-time computable* distributions $\mathcal{D}$ allow polynomial-time computable optimal compression schemes. Many natural distributions are polynomial-time computable, but there are a number of important exceptions. The output of a pseudorandom generator, for example, defines a distribution that is not optimally compressible in polynomial time and, hence, is not polynomial-time computable.

1.1.4 Decision versus search

The second result that we present, due to Ben-David *et al.* [12], shows that if $(\mathrm{BH},\mathcal{U}^{\mathrm{BH}})$ has a good-on-average algorithm, then for all NP relations R and all polynomial-time computable distributions $\mathcal{D}$, there is an efficient algorithm that, given x sampled from $\mathcal{D}$, almost always finds a y such that $R(x,y)$, provided that such a y exists. This shows that the question of whether there are intractable-on-average search problems in NP (with respect to polynomial-time computable distributions) is equivalent to the question of whether there are intractable-on-average decision problems in NP (with respect to such distributions). Both questions are equivalent to the specific decision problem $(\mathrm{BH},\mathcal{U}^{\mathrm{BH}})$ being intractable.

1.1.5 Computable, samplable, and arbitrary distributions

The restriction of the completeness result to samplable distributions is quite undesirable because it rules out reasonably natural distributions that can occur in certain applications. Ideally, it would be desirable that the theory put no restriction whatsoever on the distributions, and that we could prove results of the form "if there is a good-on-average algorithm for $(\mathrm{BH},\mathcal{U}^{\mathrm{BH}})$, then for every L in NP and every distribution $\mathcal{D}$ there is a good-on-average algorithm for $(L,\mathcal{D})$." The conclusion, however, is equivalent to $\mathrm{P} = \mathrm{NP}$.[4] More specifically, there is a distribution $\mathcal{D}$ such that, for every language L in NP, if there is a good-on-average algorithm for $(L,\mathcal{D})$ then there is an efficient worst-case algorithm for L. As we discuss below, there are difficulties in relating the worst-case complexity to the average-case complexity of all problems in NP, and so it seems unlikely that the theory can be generalized to handle completely arbitrary distributions. An important intermediate case between polynomial-time computable distributions and arbitrary distributions is the class of *polynomial-time samplable distributions*. This class includes some natural distributions that are not polynomial-time computable (e.g., the output of a pseudorandom generator), and an

[4] This was first proved by Levin. In Section 2.5 we present a later proof by Li and Vitányi [55].

argument can be made that any distribution that occurs "in nature" should be samplable. Impagliazzo and Levin [42] show that the completeness result can be extended to all samplable distributions. That is, if (BH,$\mathcal{U}^{\mathrm{BH}}$) admits a good-on-average algorithm, then for every problem L in NP and every samplable distribution $\mathcal{D}$, the problem $(L,\mathcal{D})$ has a good-on-average algorithm. In Sections 5.1 and 5.2, we present two proofs of this result. A simpler one, appearing in the article of Impagliazzo and Levin, which applies only to some (but not all) definitions of "good-on-average," and a second proof, also due to Impagliazzo and Levin, but unpublished, that is more complex but that applies to all definitions. The first proof is similar to the proof of the completeness result for polynomial-time computable distributions, but using a randomized encoding scheme. An input x for L is mapped into an input $(M',(r,C(r,x)),1^t)$ for BH, where r is randomly chosen. The desired properties of the randomized encoding C are (i) over the choices of r, the encoding $x \to (r,C(x,r))$ is "approximately injective," and (ii) the distribution $(r,C(x,r))$ is "approximately uniform" when r is uniformly chosen and x is sampled from $\mathcal{D}$. Some additional difficulties arise: in order to compute the randomized encoding one needs some extra information about x, and the reduction just "guesses" all possible values for this extra information, and, for technical reasons, this forces us to work with the *search* rather than the *decision* version of L. This is done without loss of generality given the reduction of Ben-David *et al.* [12]. The idea for the second proof is that, if S is the sampling algorithm for L, and L is hard-on-average over the outputs of S, then the problem "on input r, is it true that $S(r) \in L$?" should be hard-on-average with respect to the uniform distribution. This intuition is quite difficult to translate into a proof, especially in the case in which the computation of the sampler S is a one-way function.

1.1.6 Worst case versus average case

In order to unify the theory of average-case complexity with the rest of complexity theory, it would be highly desirable to prove a theorem of the form, "if P $\neq$ NP then there is a hard-on-average problem $(L,\mathcal{D})$, where L is in NP and $\mathcal{D}$ is samplable." In order to prove such a result

via a reduction, we would need to find an oracle algorithm R (the reduction) such that if A is a good-on-average algorithm for $(L, \mathcal{D})$, then R^A is a worst-case efficient algorithm for, say, 3SAT. Feigenbaum and Fortnow [27] show that (under standard assumptions) such a result cannot be proved via a *non-adaptive random reduction*, that is, via an algorithm R that makes non-adaptive queries and such that each query has the distribution $\mathcal{D}$ (regardless of the input of R). Bogdanov and Trevisan [15] show that the same impossibility result holds even if R is allowed to make arbitrary non-adaptive queries, provided that R works for arbitrary oracles. It remains possible that a worst-case-to-average-case reduction in NP exists which makes *adaptive* access to the oracle, or that uses the *code* of the algorithm A (and, hence, does not work for arbitrary oracles). Guttfreund and Ta-Shma [37] make some progress in the latter direction. An even more ambitious goal is to show, via reductions, that "if P $\neq$ NP then one-way functions exist." The result of Bogdanov and Trevisan rules out the possibility of proving such a result via oracle non-adaptive reductions; Akavia *et al.* [9] present a simpler proof in the setting of one-way functions (which, unlike the Bogdanov-Trevisan proof, works also in the uniform setting) and are also able, for a restricted class of one-way functions, to rule out non-adaptive reductions.

1.1.7 Degrees of average-case intractability

If a problem L is worst-case intractable, then every efficient algorithm makes an infinite number of mistakes; if a problem $(L, \mathcal{D})$ is average-case intractable, then every efficient algorithm makes mistakes[5] on a set of inputs that has noticeably large probability according to $\mathcal{D}$. Given the difficulties in relating these two settings, it is interesting to ask what happens if we consider different quantitative formulations of "noticeably large." O'Donnell [61] shows that any quantification between $1/2 - 1/n^{.33}$ and $1/\text{poly}(n)$ leads essentially to an equivalent intractability assumption. O'Donnell's argument, presented in Chapter 6, gives a far-reaching generalization of Yao's XOR Lemma [76].

[5] Or *fails*, depending on the definition of average-case tractability that we are using.

1.1.8 Specific problems

Eventually, we would like the theory to talk about the complexity of specific natural problems with specific natural distributions. It follows from Cook's reduction that if there is a hard-on-average problem $(L, \mathcal{D})$, where L is in NP and $\mathcal{D}$ is samplable, then every NP-hard problem is hard on average with respect to some samplable distribution, albeit a very unnatural one. On the other hand, Levin's completeness result shows (under the same assumption) that there are hard-on-average problems $(L, \mathcal{D})$, where $\mathcal{D}$ is uniform, but L is quite artificial. Yet, the theory of average-case completeness has little to say about specific cases of interest where both L and $\mathcal{D}$ are natural: for instance, the hardness of 3SAT or maximum independent set with respect to natural distributions on inputs.

A specific problem whose average-case behavior has been widely investigated is random kSAT with respect to the following distribution of instances: Choose at random $m_k(n)$ out of the $2^k\binom{n}{k}$ possible clauses of kSAT independently. The tractability of this problem appears to depend heavily on the number of clauses $m_k(n)$. While it is believed that random kSAT is hard for certain choices of $m_k(n)$, no hardness result supporting this intuition is known. However, Feige [23] shows the following surprising connection between hardness of random 3SAT and hardness of approximation: Assuming that random 3SAT is hard for certain values of $m_3(n)$, it is *worst-case hard* to approximate certain problems in NP (e.g., maximum bipartite clique within $n^{-\varepsilon}$ for some $\varepsilon > 0$.)

For certain *lattice problems* we know an equivalence between worst-case and average-case complexity [5, 57, 59, 64]. If such equivalences could be proved for NP-complete lattice problems, we would have a positive solution to the question of whether the existence of hard-on-average problems in NP can be based on the worst-case hardness of NP-complete problems.

1.2 A historical overview

In this section we review the historical progression toward the results described in the previous section.

1.2.1 One-way functions and cryptography

The average-case performance of algorithms on random inputs has been studied since the beginning of the modern theory of efficient algorithms in the 1950s and 1960s. Such work was often focused on problems for which worst-case polynomial-time algorithms were also known. The third volume of *The art of computer programming* [49] (published in 1973) extensively surveys average-case analyses of algorithms for problems such as sorting and median finding.

The study of the average case of (conjectured) intractable problems began in the 1970s motivated by the development of the foundations of cryptography and by interest in heuristic approaches to NP-complete problems.

When Diffie and Hellman [20] introduced the notion of public key cryptography, they speculated that one could base a trapdoor permutation on the difficulty of an NP-complete problem.[6] Even, Yacobi and Lempel [22,51] devised a public key cryptosystem such that an efficient adversary that breaks the system *for every key* implies an efficient algorithm for an NP-complete problem. An efficient adversary that breaks the system *on almost all keys*, however, is also discussed.

Shamir [68] discusses the difficulty in formulating a definition of intractability for cryptographic applications. Worst-case complexity is immediately seen as inadequate. Furthermore, Shamir emphasizes that a cryptographic system cannot be considered secure if there is an attack that takes expected polynomial time. In fact, Shamir adds, it is not even enough to rule out expected polynomial time attacks. Consider, for example, a system that can be broken by an attacker whose *expected* running time is very large but whose *median* running time is efficient. This is possible if the attacker takes a very long time, say, on one-third of the keys but is efficient otherwise. Even though the expected running time of the adversary is large, such a system cannot be considered secure.

[6] Indeed, Diffie and Hellman give two main justifications for their claim that "we stand on the brink of a revolution in cryptography": the availability of cheap and efficient computers (in the 1970s!) and the development of NP-completeness.

The median running time of an adversary is thus a better complexity measure of the expected running time, Shamir notes, but one needs to go beyond, and consider the running time of, say, the 1% fraction of inputs on which the algorithm is fastest. This short discussion anticipates the formal definition of one-way function and the difficulties in defining a robust notion of "average-case tractability" in Levin's theory of average-case complexity.

The work of Blum, Goldwasser, Micali, and Yao [14,35,76] put cryptography on solid foundational grounds, and introduced the modern definitions of one-way functions, trapdoor permutations, pseudorandom generators, and secure encryption. In their definition, an efficiently computable function f is one-way if there is no polynomial-time algorithm that finds a pre-image of $f(x)$ with more than inverse polynomial probability over the choice of x. This means that if f is a one-way function, then the computational problem "given $y = f(x)$ find a pre-image of y," has no algorithm of expected polynomial time, no algorithm of median polynomial time, no algorithm that runs in polynomial time on the easiest 1% fraction of inputs, and so on.

1.2.2 Levin's theory of average-case intractability

The development of the theory of NP-completeness gave evidence that a large number of important computational problems do not admit worst-case efficient algorithms and motivated the design of good-on-average algorithms as a way to "cope" with intractability.

Following this approach, the goal is to analyze worst-case superpolynomial-time algorithms for NP-complete problems and to show that on "typical" instances they are efficient. A celebrated example is Karp's algorithm for TSP in the plane [46]. An annotated bibliography by Karp *et al.* [47] written in 1985 reports several results on average-case tractability of NP-complete problems on natural distributions.

The initial success in the design of good-on-average algorithms led to the question of the limitations of such an approach. Are there NP-complete problems that, with respect to natural distributions, do not even have good-on-average algorithms? Are there general techniques,

analogous to the theory of NP-completeness, to prove average-case intractability?[7]

Levin [53] laid the foundations for a theory of the average-case tractability of problems in NP. He introduced the definition of average-case tractability and of reduction outlined above and proved the first completeness result, for the class $(\mathrm{NP}, \mathrm{PComp})$ of problems $(L, \mathcal{D})$ such that L is in NP and $\mathcal{D}$ is polynomial-time computable.

Levin's article, both in the one-page conference version and in the two-page full version [53], gives few details about the intuition behind the definitions and the possibility of generalized or alternative definitions.

Ben-David *et al.* [12] consider two issues not addressed in Levin's article. One issue is the class of distributions to consider. Levin restricts his attention to the class of "polynomial time computable distributions" that includes several natural distributions but that excludes, for example, the output of a pseudorandom generator and other natural distributions. Ben David et al. observe that the more general class of "efficiently samplable" distributions is a better formalization of the notion of natural distribution and formulate the question of whether Levin's completeness result can be extended to the corresponding class $(\mathrm{NP}, \mathrm{PSamp})$ of distributional problems $(L, \mathcal{D})$ such that L is in NP and $\mathcal{D}$ is samplable. Another issue studied in [12] is the average-case complexity of *decision* versus *search* problems, and their main result shows that if every decision problem in NP can be solved efficiently with respect to the uniform distribution, then every search problem in NP can also be solved efficiently with respect to the uniform distribution. Impagliazzo and Levin [42], solving the main open question formulated in [12], prove that there is a problem that is complete for $(\mathrm{NP}, \mathrm{PSamp})$.

[7] Interestingly, around the same time (mid-1970s), another approach was studied to "cope" with the intractability of NP-complete optimization problems, namely, to design provably efficient *approximate* algorithms that deliver near-optimal solutions, and the question was asked of when not even such algorithms exist. In the 1990s, the theory of probabilistically checkable proofs gave a powerful tool to prove intractability of approximation problems. A satisfactory and general theory to prove average-case intractability, unfortunately, does not exist yet.

1.2.3 Average-case intractability and derandomization

Yao [76] proves that the existence of pseudorandom generators implies the possibility of derandomizing probabilistic algorithms, and that pseudorandom generators can be constructed using one-way permutations. (Håstad *et al.* [39] later proved that the existence of one-way functions is sufficient.) The existence of a one-way permutation f can be stated as the average-case intractability of the *distributional search problem* of inverting f on a random input, so Yao's result proves that a specific average-case assumption (for certain search problems within NP) implies derandomization of probabilistic algorithms. The connection between average-case complexity and derandomization became more direct, simpler, and more general in the work of Nisan and Wigderson [60]. Their work requires the existence of hard-on-average distributional *decision* problems in EXP. The work of Nisan and Wigderson raised the question of whether derandomization could be based on *worst-case* assumptions about problems in EXP instead of average-case assumptions. The question led to the study of worst-case versus average-case complexity in EXP, and to such tools as *random self-reduction* [10], *amplification of hardness* [41, 44], and *error-correcting codes* [69]. As a result of this decade-long investigation, we now know that worst-case and average-case are equivalent in complexity classes such as EXP and PSPACE. The interested reader can find an account of such results in survey articles by Trevisan [71] (see, in particular, Chapter 4) and by Kabanets [45].

1.2.4 Worst-case versus average case within NP

The proofs of the worst-case and average-case equivalence for complete problems in EXP, PSPACE, and other classes raise the question of whether a similar worst-case and average-case equivalence also holds for intractable problems within NP. This is related to fundamental questions in the foundations of cryptography: Is it possible to base one-way functions on NP-completeness? If so, what about one-way permutations or public key encryption?

On the one hand, it is easy to see that one-way permutations cannot be based on NP-completeness, unless NP = coNP (or AM = coAM,

if one allows randomized reductions, or NP/poly = coNP/poly, if one allows non-uniform reductions). Not even the intractability of *worst-case* inversion can be based on NP-completeness (see Section 7.2).

On the other hand, it is possible to define "one-way functions" that are computable in polynomial time and that cannot have a "worst-case inverter" (i.e., a polynomial time inverter that works on all inputs) unless P = NP. For this reason, when we ask whether the existence of one-way functions (under the standard, average-case, definition) can be based on NP-completeness, we are asking a question about the *average-case complexity* of inverters.

To clarify before we continue: The existence of one-way permutations implies the existence of one-way functions, which implies the existence of hard-on-average distributional problems in $(\mathrm{NP}, \mathrm{PSAMP})$,[8] which implies that P is different from NP. We do not know how to prove the inverse of any of those implications, even though we believe that all the statements are true, and so they all imply each other vacously.

We can ask, however, whether reverse implications can be proved via *reductions*, that is, for example, whether there is a distributional problem $(L, \mathcal{D})$ in $(\mathrm{NP}, \mathrm{PSAMP})$ and a reduction R such that, for every algorithm A that solves $(L, \mathcal{D})$ well on average, the reduction R plus the algorithm A give a worst-case algorithm for 3SAT.

Feigenbaum and Fortnow [27] study a special case of the above question. They consider the case in which R is a "non-adaptive random self-reduction." They show that the existence of such a reduction implies the collapse of the polynomial hierarchy (which contradicts standard conjectures). The result of Feigenbaum and Fortnow rules out a certain way of proving equivalence of worst-case and average-case for NP-complete problems, including the way used in the work on EXP and PSPACE [10, 41, 44, 69] (see Section 7.3).

In a celebrated breakthrough, Ajtai [5], describes a distributional problem in $(\mathrm{NP}, \mathrm{PCOMP})$ whose average-case complexity is at least as high as the worst-case complexity of a related (promise) problem in NP–a version of the shortest vector problem for lattices in $\mathbb{R}^n$. Ajtai also proves the existence of one-way functions that are based on the

[8] This implication is non-trivial; see Section 4.3.

worst-case complexity of problems in NP. Ajtai and Dwork [7] present a public key cryptosystem based on a worst-case assumption, and Micciancio and Regev [57, 59, 64] present various improvements.

The security of the cryptosystems of Ajtai, Dwork, Micciancio, and Regev relies on the worst-case complexity of problems that are not known to be NP-complete and, in fact, are in NP $\cap$ coNP. It remains an open question whether these techniques can be refined and improved to the point where cryptography primitives can be constructed that rely on the worst-case complexity of an NP-complete problem.

Bogdanov and Trevisan [15] prove that no non-adaptive worst-case to average-case reduction exists for NP-complete problems unless NP/poly = coNP/poly. Akavia *et al.* [9] prove that one-way functions cannot be based on NP-complete problems via non-adaptive reductions unless AM = coAM (see Section 7.3).

It seems likely that reductions cannot relate worst-case and average-case hardness in NP. What about different degrees of average-case intractability? For instance, if there exist distributional problems in NP that are hard on some non-negligible fraction of instances, does it follow that there are distributional problems in NP that are hard on almost all instances? These questions have been answered in the affirmative by O'Donnell [61] and Healy, Vadhan, and Viola [40] in the non-uniform setting and by Trevisan [70, 72] in the uniform setting (see Chapter 6).

2

Definitions of "Efficient on Average"

A *distributional decision problem* is a pair $(L, \mathcal{D})$, where L is a language and $\mathcal{D}$ describes how inputs are distributed. There are various possible formalizations of how $\mathcal{D}$ is specified, of what constitutes a "natural" subset of distribution of inputs to restrict to, and of what it means for a distributional problem to have a good-on-average algorithm. We discuss the various definitions, and the relations among them, in this section.

2.1 Distribution over inputs

There are at least two common conventions on how to specify $\mathcal{D}$. The convention introduced by Levin [53] is that $\mathcal{D}$ is a probability distribution over the set $\{0,1\}^*$ of all possible bit strings. This convention is convenient in many applications and, for example, it leads to a simple definition of reduction preserving average-case algorithms. Sometimes, however, the single-distribution convention leads to counterintuitive definitions: in the *uniform distribution* over $\{0,1\}^*$, as defined by Levin, each binary string of length n has probability $\Theta(n^{-2}2^{-n})$. In the single-distribution setting it is also harder to quantify average-case hardness and to give definitions of circuit complexity, and both of these notions are important for applications to derandomization.

The other possibility is to define for each n a finite distribution D_n, with the intuition that D_n is a distribution over inputs of "size" n, and to let $\mathcal{D}$ be the *ensemble* $\mathcal{D} = \{D_n\}_{n>0}$. This convention is common in cryptography and derandomization. In cryptography, it is common to call n the *security parameter* of the distribution D_n.

In this article we adopt the second convention, where $\mathcal{D}$ is an ensemble of distributions. When discussing average-case complexity with respect to samplable ensembles, the two definitions are essentially equivalent, as we show in Appendix A.

In Section 3 we discuss an average-case analog of the notion of NP-completeness. Intuitively, we would like a definition of "average-case NP-hard" distributional problem $(L, \mathcal{D})$ such that if $(L, \mathcal{D})$ is average-case tractable then for every problem L' in NP and every ensemble $\mathcal{D}'$, the distributional problem $(L', \mathcal{D}')$ is also average-case tractable. Unfortunately, such an approach is unlikely to work:

(1) As we show in Section 2.5, a conclusion of the form "for every problem L' in NP and every $\mathcal{D}'$, the distributional problem $(L', \mathcal{D}')$ is average-case tractable" implies P = NP, even if we allow very weak notions of average-case tractability.
(2) As we show in Chapter 7, it is unlikely that we can use reductions to prove statements of the form "if $(L, \mathcal{D})$ is average-case tractable then P = NP," where L is in NP and $\mathcal{D}$ is, say, the uniform ensemble.

Together, these two results imply that an average-case analog of the theory of NP-completeness cannot refer to the class of all distributional problems $(L, \mathcal{D})$ with L in NP, and that it is necessary on put some restriction on the class of distributions to be considered.

The most natural restriction is to consider *samplable* ensembles, that is, ensembles of distributions that can be realized as outputs of a polynomial-time sampling algorithm. There are, in turn, several possible formalizations of the notion of samplable distributions: among other choices, we may require the sampling algorithm to *always* run in polynomial time (in which case the sampler is said to run in *strict polynomial time*) or to run in *expected* polynomial time (the latter

notion itself has various possible formalizations), and we may require the output of the sampler to be a *perfect*, *statistical*, or *computational* simulation of the true distribution. The distinction between these various notions of efficient samplability is important in the study of *zero-knowledge* protocols, and we refer the reader to the chapter on Zero Knowledge in Oded Goldreich's book [31]. For our purposes, it will be convenient to just consider the simplest definition, corresponding to perfect sampling with strict polynomial running time.[1]

Definition 2.1. (samplable ensemble) An ensemble $\mathcal{D} = \{D_n\}$ is *polynomial-time samplable* if there is a randomized algorithm A that, on input a number n, outputs a string in $\{0,1\}^*$ and:

- there is a polynomial p such that, on input n, A runs in time at most $p(n)$, regardless of its internal coin tosses;
- for every n and for every $x \in \{0,1\}^*$, $\mathbf{Pr}[A(n) = x] = D_n(x)$.

We will also be interested in a more restricted class of distributions, those for which the *cumulative* probability of a given string is efficiently computable. Let $\preceq$ denote the lexicographic ordering between bit strings, then if D is a distribution we define

$$f_D(x) = D(\{y : y \preceq x\}) = \sum_{y \preceq x} D(y).$$

Definition 2.2. (computable ensemble) We say that an ensemble $\mathcal{D} = \{D_n\}$ is *polynomial-time computable* if there is an algorithm that, given an integer n and a string x, runs in time polynomial in n and computes $f_{D_n}(x)$.

Observe that if $\{D_n\}$ is a computable ensemble, then in particular the function $D_n(x)$ is computable in time polynomial in n.

We let PSamp denote the class of polynomial-time samplable ensembles, and PComp denote the class of polynomial-time computable ensembles.

[1] We stress, however, that the results that we prove about samplable ensembles remain true even if we adopt more relaxed definitions of samplability.

The *uniform ensemble* $\mathcal{U} = \{U_n\}$, where U_n is the uniform distribution over $\{0,1\}^n$, is an example of a polynomial-time computable ensemble. Abusing notation, we also denote the class whose only member is the uniform ensemble by $\mathcal{U}$.

It is not difficult to see that every polynomial-time computable ensemble is also polynomial-time samplable (see Section 3.3). The converse does not hold unless $\mathrm{P} = \mathrm{P}^{\#\mathrm{P}}$. In fact, $\mathrm{PComp} = \mathrm{PSamp}$ if and only if $\mathrm{P} = \mathrm{P}^{\#\mathrm{P}}$.

Distributional complexity classes: A *distributional complexity class* is a collection of distributional decision problems. For a class of languages $\mathbf{C}$ and a class of ensembles $\mathfrak{D}$, we use $(\mathbf{C}, \mathfrak{D})$ to denote the distributional complexity class consisting of all problems $(L, \mathcal{D})$, where $L \in \mathbf{C}$ and $\mathcal{D} \in \mathfrak{D}$. In this study we focus on the distributional complexity classes $(\mathrm{NP}, \mathrm{PSamp})$, $(\mathrm{NP}, \mathrm{PComp})$, and $(\mathrm{NP}, \mathcal{U})$.

2.2 Heuristic and errorless algorithms

In this section we define two notions of average-case tractability.

Suppose that we are interested in algorithms that are efficient on average for some samplable ensemble $\mathcal{D} = \{D_n\}$. For technical reasons, our algorithms are given, in addition to the input x, a parameter n corresponding to the distribution D_n from which x was sampled. We write $A(x;n)$ to denote the output of algorithm A on input x and parameter n.

2.2.1 Average polynomial time and errorless heuristics

We begin by considering algorithms that never make mistakes and that are efficient on "typical instances." A simple measure of average-case complexity of an algorithm A would be its expected running time, and so we may think of defining an algorithm A as having "polynomial on average" running time for a distributional problem $(L, \mathcal{D})$ if there is a polynomial p such that

$$\mathbf{E}_{x \sim D_n}[t_A(x;n)] = \sum_{x \in \{0,1\}^*} D_n(x) t_A(x;n) \leq p(n),$$

for every n, where $t_A(x;n)$ is the running time of A on input x and parameter n.

Such a definition is problematic because there are algorithms that we would intuitively consider to be "typically efficient" but whose expected running time is superpolynomial. For example, suppose that A is an algorithm of expected polynomial running time, and let B be an algorithm that is quadratically slower than A (i.e., for every x, $t_B(x;n) = t_A(x;n)^2$). Then we should definitely think of B as being typically efficient. Suppose, however, that D_n is the uniform ensemble and that A runs in time, say, $O(n^2)$ on all inputs of length n, except on a set of $2^{n/2}$ inputs on which it takes time $O(2^{n/2})$; then the expected running time of A is $O(n^2)$ (the few "hard inputs" only contribute an additive constant to the average running time). If B, however, is quadratically slower than A, then B takes time $O(n^4)$ on all inputs except on $2^{n/2}$ on which it takes time $O(2^n)$. The average expected running time of B is now $O(2^{n/2})$, dominated by the time taken on the hard inputs.

In order to be less dependent on the running time of exceptional inputs, we may decide to look at the *median* running time instead of the expected running time. Such a choice would work well with the above example: both A and B have polynomial median running time. More generally, if A is an algorithm of polynomial median running time and B runs polynomially slower than A, then B must also have polynomial median running time.

Consider, however, an algorithm that runs in time $O(n^2)$ on $\frac{2}{3} \cdot 2^n$ inputs and in time $O(2^n)$ on $\frac{1}{3} \cdot 2^n$ inputs of length n. Such an algorithm has polynomial median running time with respect to the uniform ensemble, but intuitively we would not consider it to be a "typically" efficient algorithm.

We may choose to consider the 99th percentile instead of the median, but such a threshold would be arbitrary. What we would really like to capture with a definition is the notion that a "typically efficient" algorithm may take very long, even exponential time, on some inputs, but that the fraction of inputs requiring larger and larger running time is smaller and smaller. In formalizing this intuition, it is natural to require a *polynomial trade-off* between running time and fraction of inputs. This leads us to our first definition.

Definition 2.3. (average polynomial running time – trade-off definition) An algorithm A has average polynomial running time with respect to the ensemble $\mathcal{D}$ if there is an $\varepsilon > 0$ and a polynomial p such that for every n and every t:

$$\mathbf{Pr}_{x \sim D_n}[t_A(x;n) \geq t] \leq \frac{p(n)}{t^{\varepsilon}}.$$

If A satisfies the above definition, then the median running time of A is polynomial, and, furthermore, A runs in polynomial time on all but at most a $1/n$ fraction of the inputs, in time at most $n^{O(\log n)}$ on all but at most a $1/n^{\log n}$ fraction of the inputs, and so on. Levin gave the following equivalent definition.

Definition 2.4. (average polynomial running time – Levin's definition) An algorithm A has average polynomial running time with respect to the ensemble $\mathcal{D}$ if there is an $\varepsilon > 0$ such that

$$\mathbf{E}_{x \sim D_n}[t_A(x;n)^{\varepsilon}] = O(n).$$

Naturally, $O(n)$ can be replaced by an arbitrary polynomial in n. The two definitions are easily seen to be equivalent.

Proposition 2.5. An algorithm A has average polynomial running time with respect to the ensemble $\mathcal{D}$ according to Definition 2.3 if and only if it does according to Definition 2.4.

Proof. Suppose that the running time t_A of A satisfies

$$\mathbf{Pr}_{D_n}[t_A(x;n) \geq t] \leq n^c t^{-\varepsilon},$$

for some constants c, ε and for every sufficiently large n. Define $\delta = \varepsilon/(c+2)$. Then,

$$\begin{aligned}\mathbf{E}_{D_n}[t_A(x;n)^{\delta}] &= \sum_t \mathbf{Pr}_{D_n}[t_A(x;n)^{\delta} \geq t] \\ &\leq n + \sum_{t \geq n} \mathbf{Pr}_{D_n}[t_A(x;n) \geq t^{1/\delta}]\end{aligned}$$

$$\begin{aligned}
&\leq n + \sum_{t \geq n} n^c t^{-\varepsilon/\delta} \\
&= n + \sum_{t \geq n} n^c t^{-(c+2)} \\
&\leq n + \sum_{t} t^{-2} \\
&= n + O(1).
\end{aligned}$$

This proves that if A satisfies Definition 2.3 then it satisfies Definition 2.4. For the other implication, suppose

$$\mathbf{E}_{D_n}[t_A(x;n)^\varepsilon] = O(n).$$

Then, by Markov's inequality

$$\mathbf{Pr}_{D_n}[t_A(x;n) \geq t] = \mathbf{Pr}_{D_n}[t_A(x;n)^\varepsilon \geq t^\varepsilon] \leq \frac{\mathbf{E}_{D_n}[t_A(x;n)^\varepsilon]}{t^\varepsilon} = O(nt^{-\varepsilon}).$$

□

We now describe a third equivalent way to think of average polynomial time. Suppose that A is an algorithm of average polynomial running time according to the above definitions. If we think about running A "in practice," it is reasonable to assume that we will not be able to run A for more than a polynomial number of steps. We can then think of the inputs on which A takes superpolynomial time as inputs on which A "fails," because we have to stop the computation without being able to recover the result.

The notion of an algorithm that fails on some inputs is captured by the following definition.

Definition 2.6. (errorless heuristic scheme) We say that an algorithm A is a (*fully polynomial-time*) *errorless heuristic scheme* for $(L, \mathcal{D})$ if there is a polynomial p such that

(1) For every $n, \delta > 0$, and every x in the support of D_n, $A(x; n, \delta)$ outputs either $L(x)$ or the special failure symbol $\perp$.
(2) For every $n, \delta > 0$, and every x in the support of D_n, $A(x; n, \delta)$ runs in time at most $p(n/\delta)$.

(3) For every n and every $\delta > 0$,

$$\mathbf{Pr}_{x \sim D_n}[A(x; n, \delta) = \perp] \leq \delta.$$

We now show that errorless heuristic schemes are yet another way of capturing the notion of average-case tractability of Definitions 2.3 and 2.4.

Proposition 2.7. A distributional problem $(L, \mathcal{D})$ admits a fully polynomial-time errorless heuristic scheme if and only if it admits an algorithm whose running time is *average-polynomial* according to Definitions 2.3 and 2.4.

Proof. Suppose that A is an algorithm that runs in average-polynomial time according to Definition 2.3, that is, assume that there is a polynomial p and an $\varepsilon > 0$ such that for every n,

$$\mathbf{Pr}_{D_n}[t_A(x; n) \geq t] \leq \frac{p(n)}{t^{\varepsilon}}.$$

Then define the algorithm A' that on input x and parameters n, δ simulates $A(x; n)$ for $(p(n)/\delta)^{1/\varepsilon}$ steps. If the simulation halts within the required number of steps, then $A'(x; n, \delta)$ gives the same output as $A(x; n)$; otherwise $A'(x; n, \delta)$ outputs $\perp$. It is easy to see that A' satisfies the definition of an errorless heuristic scheme.

Suppose now that A' is an errorless heuristic scheme for $(L, \mathcal{D})$. Define the algorithm A as follows: On input $(x; n)$, simulate $A(x; n, 1/2)$, if $A(x; n, 1/2) \neq \perp$, then return the output of $A(x; n, 1/2)$, otherwise simulate $A(x; n, 1/4)$, and so on, simulating $A(x; n, 1/8), \ldots, A(x; n, 2^{-k}), \ldots$ until we reach a value of δ such that $A(x; n, \delta) \neq \perp$. Eventually, the algorithm succeeds because when $\delta < D_n(x)$ then $A(x; n, \delta)$ cannot output $\perp$. After k iterations, A uses time $\sum_{i=1}^{k} p(2^i n) = O(k \cdot p(2^k n))$, for a polynomial p, and it halts within k iterations on all but a $1/2^k$ fraction of inputs. It is now easy to verify that A runs in average polynomial time according to Definition 2.3. □

Having given three equivalent formulations of "efficient on average" algorithms, we are ready to define a complexity class of distributional problems.

Definition 2.8. (average polynomial time) We define AvgP to be the class of distributional problems that admit an errorless heuristic scheme.

The third approach to the definition leads naturally to a finer quantitative definition.

Definition 2.9. (errorless heuristic algorithms) Let L be a language, $\mathcal{D}$ be an ensemble, and $\delta : \mathbb{N} \to \mathbb{R}^+$. We say that an algorithm A is an *errorless heuristic algorithm* for $(L, \mathcal{D})$ with *failure probability* at most δ if

- For every n and every x in the support of D_n, $A(x; n)$ outputs either $L(x)$ or the special failure symbol $\perp$, and
- For every n, $\mathbf{Pr}_{x \sim D_n}[A(x; n) = \perp] \leq \delta(n)$.

For a function $t : \mathbb{N} \to \mathbb{N}$, we say that $(L, \mathcal{D}) \in \mathrm{Avg}_\delta\mathrm{DTIME}(t(n))$ if there is an errorless heuristic deterministic algorithm A that for every n and every $x \in \mathrm{Supp}(D_n)$ runs in time $t(n)$ with failure probability at most $\delta(n)$.

We define $\mathrm{Avg}_\delta\mathrm{P}$ as the union over all polynomials p of $\mathrm{Avg}_\delta\mathrm{DTIME}(p(n))$.

We use $\mathrm{Avg}_{\mathrm{neg}}\mathrm{P}$ to denote the union of all classes $\mathrm{Avg}_\delta\mathrm{P}$, where δ is a negligible function. Recall that δ is negligible if, for every polynomial p and for every sufficiently large n, $\delta(n) \leq 1/p(n)$.

Observe that an errorless heuristic scheme for a distributional problem automatically yields errorless heuristic algorithms with error probability $1/p(n)$ for the same problem, for every polynomial p. For certain problems, heuristic algorithms can conversely be turned into heuristic schemes. We discuss this connection in Section 3.3.

2.2.2 Heuristic algorithms

So far we have considered only algorithms that never make mistakes: they always either produce a correct answer or fail. It is also interesting to consider algorithms that return incorrect answers on a small fraction of inputs.

Definition 2.10. (heuristic algorithms) Let L be a language, $\mathcal{D}$ be an ensemble, and $\delta : \mathbb{N} \to \mathbb{R}^+$. We say that an algorithm A is a *heuristic algorithm* for $(L, \mathcal{D})$ with *error probability* at most δ if for all $n > 0$,

$$\mathbf{Pr}_{x \sim D_n}[A(x; n) \neq L(x)] \leq \delta(n) \ .$$

Definition 2.11. (heuristic polynomial time) For functions $t : \mathbb{N} \to \mathbb{N}$ and $\delta : \mathbb{N} \to \mathbb{R}^+$, we say that $(L, \mathcal{D}) \in \mathrm{Heur}_\delta\mathrm{DTIME}(t(n))$ if there is a heuristic deterministic algorithm A that for every n and every $x \in \mathrm{Supp}(D_n)$ runs in time $t(n)$ with failure probability at most $\delta(n)$.

We define $\mathrm{Heur}_\delta\mathrm{P}$ as the union over all polynomials p of $\mathrm{Heur}_\delta\mathrm{DTIME}(p(n))$.

We say that an algorithm A is a *(fully polynomial-time) heuristic scheme* for $(L, \mathcal{D})$ if there is a polynomial p such that

(1) For every n, for every x in the support of D_n and every $\delta > 0$, $A(x; n, \delta)$ runs in time at most $p(n/\delta)$.
(2) For $\delta > 0$, $A(\cdot; \cdot, \delta)$ is a heuristic algorithm for $(L, \mathcal{D})$ with error probability at most δ.

We define HeurP to be the class of distributional problems that admit a heuristic scheme.

We use $\mathrm{Heur}_{\mathrm{neg}}\mathrm{P}$ to denote the union of all classes $\mathrm{Heur}_\delta\mathrm{P}$, where δ is a negligible function.

An errorless algorithm can be easily turned into a heuristic algorithm by replacing the failure symbol $\perp$ by an arbitrary output. Thus, $\mathrm{Avg}\mathbf{C} \subseteq \mathrm{Heur}\mathbf{C}$ and $\mathrm{Avg}_\delta\mathbf{C} \subseteq \mathrm{Heur}_\delta\mathbf{C}$ for all classes of this type described above.

2.3 Non-uniform and randomized heuristics

We will also be interested in non-uniform and randomized heuristic algorithms.

Deterministic heuristics turn out to be an inadequate notion in much of average-case complexity, including many of the results stated

in this survey. For instance, the decision-to-search reduction of Ben-David *et al.* in Chapter 4 and the reductions of Impagliazzo and Levin from (NP, PSAMP) to (NP,$\mathcal{U}$) in Chapter 5 are both randomized, so to understand these reductions one must first define the notion of a randomized heuristic. The results on hardness amplification in Chapter 6 make use of both randomness and non-determinism.

However, the definitions of non-uniform and randomized heuristics contain some subtleties, and if the reader feels overwhelmed by definitions at this point, he or she may skip ahead to Section 2.4.

Non-uniform heuristics: For a function $s : \mathbb{N} \to \mathbb{N}$, we define $\text{Heur}_\delta\text{SIZE}(s(n))$ and HeurP/poly in the same way we define $\text{Heur}_\delta\text{DTIME}(t(n))$ and HeurP, respectively, but referring to "circuits of size $s(n)$" instead of "algorithms running in time $t(n)$." Similarly, we define the non-uniform errorless heuristic classes $\text{Avg}_\delta\text{SIZE}(s(n))$ and AvgP/poly.

A small technical point is that, when we consider a distributional problem $(L, \{D_n\})$, the inputs in the support of D_n may have different lengths. In such a case, we need to fix a convention to allow Boolean circuits to accept inputs of various lengths. Once such a convention is chosen, then, for example, $(L, \{D_n\}) \in \text{Avg}_\delta\text{SIZE}(s(n))$ means that there is a family of circuits C_n such that, for every n: (i) C_n is of size at most $s(n)$; (ii) for every x in the support of D_n, $C_n(x)$ outputs either $L(x)$ or $\perp$; (iii) $\mathbf{Pr}_{x \sim D_n}[C(x) \neq L(x)] \leq \delta(n)$.

Randomized heuristics: When defining randomized heuristic algorithms, there are two ways in which the algorithm can fail to produce a correct answer: It can run either on an input on which the heuristic fails or run on an input for which the heuristic is good but make a bad internal coin toss. It is important to keep this distinction in mind when defining randomized *errorless* heuristic algorithms. Here "errorless" refers to the choice of input and not to the internal coin tosses of the algorithm.

In particular, we allow the randomized errorless algorithm to sometimes output incorrect answers, as long as for every instance x, the fraction of random strings for which the algorithm outputs the wrong answer is small compared to the fraction of random strings for which it outputs either the right answer or $\perp$.

Definition 2.12. (randomized errorless heuristics) Let $(L, \mathcal{D})$ be a distributional problem and $\delta : \mathbb{N} \to \mathbb{R}^+$. We say that a randomized polynomial-time algorithm A is a *randomized errorless heuristic algorithm* of failure probability at most δ if, for every $n > 0$, and every x in the support of D_n,

$$\mathbf{Pr}[A(x;n) \notin \{L(x), \perp\}] \leq 1/4,$$

where the probability is taken over the coin tosses of A, and

$$\mathbf{Pr}_{x \sim D_n}\big[\mathbf{Pr}[A(x;n) = \perp] \geq 1/4\big] \leq \delta(n),$$

where the inner probability is over the internal coin tosses of A.

To see why this definition makes sense, fix an input $(x;n)$ and imagine running the algorithm k times, for some large k. If substantially more than $k/4$—say, $k/3$—of these runs return the failure symbol $\perp$, we can interpret this as a sign that the algorithm does not know the answer for x. The second condition of Definition 2.12, together with standard Chernoff-type bounds, guarantees that this will not happen for more than a $\delta(n)$-fraction of instances $x \sim D_n$ with high probability over the randomness of the algorithm.

If, on the other hand, the number of runs that return $\perp$ is smaller than $k/3$, then the first condition of Definition 2.12 guarantees that with high probability, a majority of the runs that do not output $\perp$ will output the correct answer, so we obtain the correct answer for x with high probability over the randomness of the algorithm.

This argument shows that the choice of constant $\frac{1}{4}$ is arbitrary, and any constant bounded away from $\frac{1}{3}$ can serve in the definition. In the other direction, the algorithm A' that simulates A $k = k(n)$ times satisfies:

$$\mathbf{Pr}[A'(x;n) \notin \{L(x), \perp\}] = 2^{-\Omega(k(n))} \tag{2.1}$$

and

$$\mathbf{Pr}_{x \sim D_n}\left[\mathbf{Pr}[A'(x;n) = \perp] \geq \frac{1}{2^{k(n)/100}}\right] \leq \delta(n). \tag{2.2}$$

If the constant $\frac{1}{4}$ is replaced by 0 in the first condition of Definition 2.12, we obtain the definition of *zero-error* randomized errorless heuristics.

Definition 2.13. (randomized errorless classes) We say that $(L, \mathcal{D})$ is in $\mathrm{Avg}_\delta\mathrm{BPTIME}(t(n))$ if there is a randomized errorless algorithm A of failure probability at most $\delta(n)$ and of running time at most $t(n)$ on inputs in the support of D_n. If A is zero error, we say that $(L, \mathcal{D})$ is in $\mathrm{Avg}_\delta\mathrm{ZPTIME}(t(n))$.

We define Avg_δBPP, AvgBPP, Avg_δZPP, and AvgZPP in the obvious way.

If we choose $k(n) = O(n)$ in equations (2.1) and (2.2), the probabilities over the internal coin tosses of A' can be made smaller than 2^n, and using Adleman's proof that BPP $\subseteq$ P/poly [3], we have $\mathrm{Avg}_\delta\mathrm{BPP} \subseteq \mathrm{Avg}_\delta\mathrm{P/poly}$, AvgBPP $\subseteq$ AvgP/poly, and so on.

In the case of heuristic algorithms that are allowed to make errors, the definition simplifies as we do not have to distinguish between errors owing to bad inputs and errors owing to bad internal coin tosses.

Definition 2.14. (randomized heuristics) Let $(L, \mathcal{D})$ be a distributional problem and $\delta : \mathbb{N} \to \mathbb{R}^+$. We say that a randomized algorithm A is a randomized heuristic of failure probability at most δ if for every n,

$$\mathbf{Pr}_{x \sim D_n}\big[\mathbf{Pr}[A(x; n) \neq L(x)] \geq 1/4\big] \leq \delta(n),$$

where the inner probability is over the internal coin tosses of A.

Definition 2.15. (randomized heuristic classes) We say that $(L, \mathcal{D})$ is in $\mathrm{Heur}_\delta\mathrm{BPTIME}(t(n))$ if there is a randomized errorless algorithm A of failure probability at most $\delta(n)$ and of running time at most $t(n)$ on inputs in the support of D_n. We define Heur_δBPP and HeurBPP in the obvious way.

For all classes of the type $\mathrm{Avg}_\delta\mathbf{C}$ and $\mathrm{Heur}_\delta\mathbf{C}$ defined above, we define $\mathrm{Avg}_{\mathrm{neg}}\mathbf{C}$ and $\mathrm{Heur}_{\mathrm{neg}}\mathbf{C}$ as their union over all negligible functions δ, respectively.

For the non-uniform and randomized heuristic classes, we have the standard containments $\mathrm{Avg}\mathbf{C} \subseteq \mathrm{Heur}\mathbf{C}$. For the classes of type $\mathrm{Avg}_\delta\mathbf{C}$ and $\mathrm{Heur}_\delta\mathbf{C}$ it is possible to improve the containments in the deterministic case, as the algorithm can randomly (or non-uniformly) guess the answer for $\perp$, so that $\mathrm{Avg}_\delta\mathbf{C} \subseteq \mathrm{Heur}_{\delta/2}\mathbf{C}$.

2.4 Representing inputs

Average-case complexity is more sensitive to how we encode inputs to algorithms than worst-case complexity. For instance, operations like changing the alphabet or duplicating an instance do not have much effect in most treatments of worst-case complexity, while in average-case complexity they can considerably modify the distribution on inputs.

It will, therefore, be convenient to fix an encoding for inputs that is robust for average-case reductions and algorithms. In the applications described in this study, it will be necessary to have robust representations of the following types of inputs with respect to the uniform distributions: tuples of strings, machines, and hash functions. For instance, one feature of the encodings is that a random string in the uniform distribution will represent a valid tuple or a valid hash function with non-negligible probability. It is not difficult to imagine why this is crucial for average-case algorithms. In contrast, many natural encodings of these objects that are perfectly adequate in worst-case complexity do not have these property.

We do not try to optimize our representations in any manner; we simply choose representations that will be adequate for all applications covered in this survey.

Tuples: We represent inputs to algorithms as strings in $\{0,1\}^*$. A good representation for tuples of strings (in the uniform distribution) should have the property that the probability of generating a tuple $(x_1,\ldots,x_t)$ should be roughly $2^{-(|x_1|+\cdots+|x_t|)}$. We will adopt the following convention for tuples: First, write a prefix-free encoding of the number $|x_1|$ by repeating every bit twice and ending with 01. Then write down x_1. Repeat with x_2, x_3, up to x_t. Thus, the description length of $(x_1,\ldots,x_t)$ is $2\log|x_1| + \cdots + 2\log|x_t| + |x_1| + \cdots + |x_t| + O(t)$. Alternatively, the probability of generating

$(x_1, \ldots, x_t)$ in the uniform distribution according to this representation is $(|x_1| \ldots |x_t|)^{-2} 2^{-(|x_1|+\cdots+|x_t|+O(t))}$. Observe that this representation is prefix free.

When all of the strings in the tuple have the same length, more compact representations are of course possible; such representations will be necessary for the results on hardness amplification in Chapter 6.

Machines: Sometimes the input (or a part of it) is the description of a machine. The exact way in which machines are represented is irrelevant, so we fix an arbitrary representation for machines.

Hash functions: In Chapters 4 and 5, algorithms take as part of their input a description of a hash function h. By "hash function" we mean a random instance from a family of pairwise independent hash functions mapping $\{0,1\}^m$ to $\{0,1\}^n$ for fixed m and n. To be specific, we can think of the family of affine transformations $h(x) = Ax + b$, where A is an $m \times n$ matrix, b is an n-bit vector, and the operations are over $\mathbb{Z}_2$. We represent such transformations by specifying the tuple (A, b), so that the description length is $2 \log m + 4 \log n + mn + n + O(1)$.

For a function $h : \{0,1\}^m \to \{0,1\}^n$, we use $h|_j$ (where $1 \leq j \leq n$) to denote the function that consists of the first j output bits of h. If h is a hash function, then so is $h|_j$.

We will also consider hash functions from $\{0,1\}^{\leq m}$ (the set of binary strings of length at most m) to $\{0,1\}^n$. We will identify such functions with hash functions from $\{0,1\}^{m+1}$ to $\{0,1\}^n$, where $\{0,1\}^{\leq m}$ is embedded in $\{0,1\}^{m+1}$ in the natural way: String x maps to $0^{m-|x|}1x$.

2.5 A distribution for which worst case and average case are equivalent

In this section we show that there exists a (possibly non-samplable) ensemble of distributions with respect to which worst-case and average-case tractability are equivalent notions. Thus, the study of average-case complexity with respect to *all* ensembles reduces to the study of worst-case complexity, and in this sense it is natural to consider restricted classes such as computable and samplable ensembles, as we do in the remainder of this study.

Theorem 2.16. (Levin, Li, and Vitányi) There is an ensemble $\mathcal{D}$ such that if L is a decidable language and the distributional problem $(L, \mathcal{D})$ is in $\text{Heur}_{1/n^3}\text{P}$, then $L \in \text{P}$.

We present a proof due to Li and Vitányi [55] that relies on Kolmogorov complexity.

We consider pairs (M, w), where M is a machine and w is a string. Recall that if M is ℓ bits long and w is n bits long, then (M, w) has length $\ell + n + 2\log \ell + 2\log n + O(1)$.

For a binary string x, denote $K(x)$ as the length of the shortest pair (M, w) such that M on input w outputs x. The value $K(x)$ is called the (*prefix-free*) *Kolmogorov complexity* of x.

The *universal probability distribution* $\mathcal{K}$ is defined so that the probability of a string x is $2^{-K(x)}$. Observe that $\sum_x 2^{-K(x)} \leq 1$ since the representation of (M, w) is prefix free. (In fact, $\sum_x 2^{-K(x)} < 1$ so K is technically not a probability distribution, but we can correct this by assigning, say, to the empty string ε the probability $1 - \sum_{x \neq 0} 2^{-K(x)}$.) Finally, let $\{K_n\}$ be the ensemble of distributions, where K_n is the distribution $\mathcal{K}$ conditioned on strings of length n.

It turns out that for every language L, solving L well on average with a heuristic algorithm is as hard as solving L well on the worst case.

Proof of Theorem 2.16. We use the ensemble $\{K_n\}$ defined above.

Let A be the polynomial-time heuristic algorithm that witnesses $(L, \{K_n\}) \in \text{Heur}_{1/n^3}\text{P}$. We will argue that there is only a finite number of inputs x such that $A(x; |x|) \neq L(x)$, which implies that $L \in \text{P}$.

We first need to understand the distributions K_n in the ensemble. By definition,

$$K_n(x) = \frac{2^{-K(x)}}{\sum_{y \in \{0,1\}^n} 2^{-K(y)}}$$

and we can see that $\sum_{y \in \{0,1\}^n} 2^{-K(y)} = \Omega(1/n(\log n)^2)$ because the string 0^n has Kolmogorov complexity at most $\log n + 2\log\log n + O(1)$ and so contributes at least $\Omega(1/n(\log n)^2)$ to the sum.

This implies

$$K_n(x) = O(n(\log n)^2 \cdot 2^{-K(x)}) = 2^{-K(x)+\log n+2\log\log n+O(1).}$$

Now let x be a string of length n such that $A(x;n) \neq L(x)$; since the overall probability of all such strings is at most $1/n^3$, in particular we must have $K_n(x) \leq 1/n^3$, and

$$\begin{aligned} K(x) &= \log \frac{1}{K_n(x)} - \log n - 2\log\log n - O(1) \\ &\geq 2\log n - 2\log\log n - O(1). \end{aligned} \tag{2.3}$$

Consider now the lexicographically first string x in $\{0,1\}^n$ (if any) such that $A(x;n) \neq L(x)$. Such a string can be computed by an algorithm that, given n, computes $A(x;n)$ and $L(x)$ for all strings $x \in \{0,1\}^n$ and outputs the lexicographically first x for which $A(x;n) \neq L(x)$. (Here we are using the assumption that L is decidable.) Such an algorithm proves that $K(x) \leq \log n + 2\log\log n + O(1)$, and, for sufficiently large n, this is in contradiction with (2.3).

We conclude that there can only be a finite number of input lengths on which A and L differ, and so a finite number of inputs on which A and L differ. □

3

A Complete Problem for Computable Ensembles

In this Chapter we give a definition of reduction that preserves average-case tractability and we prove the existence of a problem complete for $(\mathrm{NP}, \mathrm{PComp})$.

3.1 Reductions between distributional problems

We begin by defining an appropriate notion of reduction. Besides the usual correctness requirement for reductions in worst-case complexity, a reduction in average-case complexity must in some sense match the distributions on instances of the two problems. Namely, in a reduction from $(L, \mathcal{D})$ to $(L', \mathcal{D}')$, we want the process of sampling an instance from $\mathcal{D}$, then applying the reduction to it, roughly yields the distribution $\mathcal{D}'$.

Definition 3.1. (reduction between distributional problems) Let $(L, \mathcal{D})$ and $(L', \mathcal{D}')$ be two distributional problems. We say that $(L, \mathcal{D})$ reduces to $(L', \mathcal{D}')$, and write $(L, \mathcal{D}) \leq_{\mathrm{AvgP}} (L', \mathcal{D}')$, if there is a function f that for every n, on input x in the support of D_n and parameter n, can be computed in time polynomial in n and

(1) (Correctness) $x \in L$ if and only if $f(x;n) \in L'$.
(2) (Domination) There are polynomials p and m such that, for every n and every y in the support of $D'_{m(n)}$,

$$\sum_{x:f(x;n)=y} D_n(x) \leq p(n) D'_{m(n)}(y).$$

Part (1) of the definition is the standard requirement of mapping reductions. The intuition for part (2) is that if we sample a string x from D_n and then compute $y = f(x;n)$, we generate y with probability not much larger than if y had been sampled according to $D'_{m(n)}$.

The reduction preserves the notions of average-case tractability as defined in Chapter 2.

Lemma 3.2. If $(L,\mathcal{D}) \leq_{\mathrm{AvgP}} (L',\mathcal{D}')$ and $(L',\mathcal{D}') \in \mathbf{C}$, where $\mathbf{C}$ is one of the distributional classes $\mathrm{AvgP}, \mathrm{Avg}_{\mathrm{neg}}\mathrm{P}, \mathrm{HeurP}, \mathrm{Heur}_{\mathrm{neg}}\mathrm{P}$, $\mathrm{AvgBPP}, \mathrm{HeurBPP}, \mathrm{AvgP/poly}, \mathrm{HeurP/poly}$, then $(L,\mathcal{D}) \in \mathbf{C}$.

Proof. For concreteness, we show the case $\mathbf{C} = \mathrm{AvgP}$, but the same proof works for all the other cases. Suppose that $(L',\mathcal{D}')$ is in AvgP and let A' be the fully polynomial-time errorless heuristic scheme for $(L',\mathcal{D}')$, let f be the reduction from $(L,\mathcal{D})$ to $(L',\mathcal{D}')$, let p and m be the polynomials as in the definition of reduction.

We claim that $A(x;n,\delta) := A'(f(x;n);m(n),\delta/p(n))$ is a fully polynomial-time errorless heuristic scheme for $(L,\mathcal{D})$.

To prove the claim, we bound the failure probability of A. Let us fix parameters n and δ, and let us define B to be the set of "bad" strings y such that $A'(y;m(n),\delta/p(n)) = \perp$, and let B_m be B restricted to the support of D'_m. Observe that $D'_{m(n)}(B_{m(n)}) \leq \delta/p(n)$. Then,

$$\begin{aligned}
\mathbf{Pr}_{x\sim D_n}[A(x;n,\delta) = \perp] &= \sum_{x:f(x;n)\in B_{m(n)}} D_n(x) \\
&\leq \sum_{y\in B_{m(n)}} p(n) D'_m(y) \\
&= p(n) \cdot D'_{m(n)}(B_{m(n)}) \\
&\leq \delta.
\end{aligned}$$

This establishes the claim and proves that $(L,\mathcal{D}) \in \mathrm{AvgP}$. □

3.2 The completeness result

In this section we prove the existence of a complete problem for $(\mathrm{NP}, \mathrm{PComp})$, the class of all distributional problems $(L, \mathcal{D})$ such that L is in NP and $\mathcal{D}$ is polynomial-time computable. Our problem is the following "bounded halting" problem for non-deterministic Turing machines:

$$\mathrm{BH} = \{(M, x, 1^t) : M \text{ is a non-deterministic Turing machine that accepts } x \text{ in} \leq t \text{ steps}\}. \tag{3.1}$$

Note that BH is NP-complete: Let L be a language in NP and M be a non-deterministic Turing machine that decides L in time at most $p(n)$ on inputs of length n. Then a reduction from L to BH is simply the mapping that takes a string x of length n to the triple $(M, x, 1^{p(n)})$.

We would like to show that the distributional problem $(\mathrm{BH}, \mathcal{U}^{\mathrm{BH}})$, where $\mathcal{U}^{\mathrm{BH}} = \{U_n^{\mathrm{BH}}\}$ is the "uniform" ensemble of inputs for BH (we will get to the exact definition of this ensemble shortly) is complete for $(\mathrm{NP}, \mathrm{PComp})$. The standard reduction is clearly inadequate because, if $(L, \mathcal{D})$ is a distributional problem in $(\mathrm{NP}, \mathrm{PComp})$ and $\mathcal{D}$ is a distribution that is very far from uniform, then the triples $(M, x, 1^{p(n)})$ produced by the reduction will not be uniformly distributed.

The key idea in the reduction is to find an injective mapping C such that if x is distributed according to $\mathcal{D}$, then $C(x)$ is distributed "almost" uniformly. The reduction then maps $(x; n)$ into $(M', C(x), 1^{p'(n)})$, where M' is a machine that on input $C(x)$ computes x and then runs M on x, and where $p'(n)$ is a polynomial upper bound to the running time of M'. We will show that such a mapping exists whenever $\mathcal{D}$ is a polynomial-time computable ensemble.

Before moving on, let us define the "uniform distribution" of inputs for BH. The instances of the problem are triples $(M, x, 1^t)$, so that if the representation of M has length ℓ and x has length n, then the length of the representation of $(M, x, 1^t)$ is $2\log\ell + 2\log n + 2\log t + \ell + n + t + \Theta(1)$.

We think of the "uniform distribution" over inputs of length N as follows: we flip random bits $b_1, \ldots, b_i$ until either $i = N$ or we have generated a valid prefix-free representation (according to the above rules) of M, x. In the former case we output $b_1, \ldots, b_N$; in the latter

case we output $(M, x, 1^{N-i})$. We denote this distribution by U_N^{BH}. In U_N^{BH}, an instance $(M, x, 1^t)$ has probability $2^{-(2\log \ell + 2\log n + \ell + n + \Theta(1))}$, where ℓ is the length of the representation of M and n is the length of x. (By convention, we declare that inputs not of the proper form $(M, x, 1^t)$ are not in the language BH.)

We now prove the following completeness result.

Theorem 3.3. The distributional problem $(\mathrm{BH}, \mathcal{U}^{\mathrm{BH}})$ is complete in $(\mathrm{NP}, \mathrm{PComp})$ under the reductions of Definition 3.1.

Proof. Let $(L, \mathcal{D})$ be a distributional problem in $(\mathrm{NP}, \mathrm{PComp})$.

Claim 3.4. Suppose $\mathcal{D} = \{D_n\}$ is a polynomial-time computable distribution over x. Then there exists an algorithm $C(x)$ such that for all n, $C(x)$ runs in time polynomial in n and

(1) For every fixed n, for all x in the support of D_n, $C(x)$ is injective as a function of x, and
(2) $|C(x)| \leq 1 + \min\left\{|x|, \log \frac{1}{D_n(x)}\right\}$.

Observe that since D_n is polynomial-time computable, there exists a polynomial $m(n)$ such that no string in the support of D_n can be more than $m(n)$ bits long.

Proof. Fix an $x \in \operatorname{Supp} D_n$. If $D_n(x) \leq 2^{-|x|}$, then simply let $C(x) = 0x$, that is, 0 concatenated with x.

If, on the other hand, $D_n(x) > 2^{-|x|}$, let y be the string that precedes x in lexicographic order among the strings in $\{0,1\}^n$ and let $p = f_{D_n}(y)$ (if x is the empty string, then we let $p = 0$). Then we define $C(x; n) = 1z$. Here z is the longest common prefix of $f_{D_n}(x)$ and p when both are written out in binary. Since f_{D_n} is computable in polynomial time, so is z. C is injective because only two binary strings s_1 and s_2 can have the same longest common prefix z; a third string s_3 sharing z as a prefix must have a longer prefix with either s_1 or s_2. Finally, since $D_n(x) \leq 2^{-|z|}$, $|C(x)| \leq 1 + \log \frac{1}{D_n(x)}$. □

Let M be the non-deterministic Turing machine that, on input y, accepts if and only if there exists a string x such that $y = C(x)$ and $x \in L$. Since L is in NP, machine M can be implemented so that, on input $C(x)$, where x is of length n, M runs in time at most $q(n)$, where q is a polynomial.

We can now describe the reduction. On input x and parameter n, the reduction outputs the instance $(M, C(x), 1^{t(x)})$ of length $N(n)$; here, $N(n)$ is chosen large enough so that when $|C(x)| \leq m(n)$, we have $t(x) \geq q(n)$ (for instance, $N(n) = m(n) + q(n) + 2\log m(n) + 2\log q(n) + O(1)$ suffices).

It can be seen immediately that $x \in L$ if and only if $(M, C(x), 1^{t(x)}) \in \mathrm{BH}$. Regarding the domination condition, we observe that the reduction is injective, and so we simply need to check that for every n and $x \in \operatorname{Supp} D_n$ we have

$$D_n(x) \leq \mathrm{poly}(n) \cdot U^{\mathrm{BH}}_{N(n)}(M, C(x), 1^{t(x)}).$$

To verify the inequality, let ℓ be the length of the binary representation of M. We have

$$U^{\mathrm{BH}}_{N(n)}(M, C(x), 1^{q(n)}) = 2^{-(2\log \ell + 2\log |C(x)| + \ell + |C(x)| + \Theta(1))}.$$

We observe that $\log |C(x)| \leq \log(m(n) + 1)$ and that $|C(x)| \leq \log(1/D_n(x)) + 1$, and so

$$U^{\mathrm{BH}}_{N(n)}(M, C(x), 1^{q(n)}) \geq 2^{-(2\log \ell + \ell)} \cdot (m(n) + 1)^{-2} \cdot D_n(x) \cdot \Omega(1)$$

as desired. □

3.3 Some observations

3.3.1 Completeness of bounded halting: A perspective

The main idea in the proof of Theorem 3.3 is that it is possible to extract the randomness from samples in a computable ensemble. In the proof of Theorem 3.3, the randomness is extracted through compression: Indeed, the algorithm C compresses samples x from D_n in such a way that the output $C(x)$ is dominated by the uniform distribution.

Another possible way to extract the randomness from samples of a computable ensemble is by inversion. Namely, if one views an instance

$x \sim D_n$ as the output of some sampler S, then the problem of extracting the randomness from x can be solved by *inverting* S. More precisely, one arrives at the following question: Given x, is there an efficient procedure that produces a random r such that $S(n;r) = x$? Such a procedure would map samples of D_n to samples of the uniform distribution and can be used to reduce the distributional problem $(L,\mathcal{D})$ to some distributional problem $(L',\mathcal{U})$. This perspective leads to an alternate proof of Theorem 3.3.[1]

Alternate proof of Theorem 3.3. First, it is not difficult to see that every polynomial-time computable ensemble $\mathcal{D} = \{D_n\}$ is also polynomial-time samplable. To sample from a distribution D_n, the sampling algorithm $S(n)$ generates random bits $r_1, r_2, \ldots, r_{m(n)}$ and, using binary search, returns the lexicographically smallest x such that $f_{D_n}(x) > 0.r_1 r_2 \ldots r_{m(n)}$. Here, $m(n)$ is the running time of the algorithm that computes f_{D_n}, and we assume without loss of generality (for technical reasons) that m is injective. It is easy to check that each sample is produced with the correct probability.

Observe that the sampler S is efficiently invertible in the following sense: There exists an algorithm I that on input $x \in \mathrm{Supp}(D_n)$ runs in time polynomial in n and outputs a uniformly random $r \in \{0,1\}^{m(n)}$ conditioned on $S(n;r) = x$ (meaning that $S(n)$ outputs x when using r for its internal coin tosses). The algorithm I first determines $f_{D_n}(x)$ and $D_n(x)$ using binary search and oracle calls to f_{D_n}, then samples a $m(n)$-bit number uniformly from the interval $(f_{D_n}(x) - D_n(x), f_{D_n}(x)]$.

Now consider the language L' that contains all r such that $S(n;r) \in L$, where $|r| = m(n)$ (recall that m is injective). Then L' is an NP language, and moreover $(L,\mathcal{D})$ reduces to the distributional problem $(L',\mathcal{U})$: The reduction is implemented by the inversion algorithm I, and both the correctness and domination properties are straightforward from the definition.

Finally, consider the canonical reduction from $(L',\mathcal{U})$ to $(\mathrm{BH},\mathcal{U}^{\mathrm{BH}})$, which maps instance r of L' to instance $(M', r, 1^{q(|x|)})$ of BH, where M'

[1] The statement is actually weaker as the alternate reduction is randomized.

is a non-deterministic Turing machine for L' and $q(n)$ is the running time of M' on inputs of length n. Let ℓ denote the size of M', and $|r| = m$. Then, for an appropriate choice of N, we have

$$\begin{aligned} U_N^{\mathrm{BH}}(M', r, 1^{q(m)}) &= 2^{-(2\log\ell + 2\log m + \ell + m + \Theta(1))} \\ &= 2^{-(2\log\ell+\ell)} \cdot m^{-2} \cdot U_m(r) \cdot \Omega(1), \end{aligned}$$

and this reduction also satisfies the domination condition (as ℓ does not grow with input size). □

The two proofs of Theorem 3.3 are not that different, as the encoding function C in the original proof plays much the same role as the inverter I in the alternate proof. However, despite the somewhat artificial technical distinction, the perspectives are quite different: To "recover" the uniform ensemble from a computable ensemble $\mathcal{D}$, one may attempt either to compress $\mathcal{D}$ or to invert its sampler. Indeed, the two approaches lead to different insights and different proofs (and even somewhat different theorems) when we extend these arguments to the case of polynomial-time samplable ensembles in Chapter 5.

3.3.2 Heuristic algorithms versus heuristic schemes

When defining average-case complexity classes, we distinguished between heuristic algorithms and heuristic schemes: For heuristic algorithms, we fix a failure probability δ and require that the algorithm succeeds on all but a δ-fraction of the instances. For heuristic schemes, we require a single algorithm that works for all δ, but we allow the running time to grow as a function of $1/\delta$.

It is clear that if a distributional problem has a heuristic scheme, then it has heuristic algorithms with failure probability $\delta(n) = n^{-c}$ for every $c > 0$. In other words, for every $c > 0$, $\mathrm{HeurP} \subseteq \mathrm{Heur}_{n^{-c}}\mathrm{P}$, $\mathrm{HeurBPP} \subseteq \mathrm{Heur}_{n^{-c}}\mathrm{BPP}$, $\mathrm{AvgP} \subseteq \mathrm{Avg}_{n^{-c}}\mathrm{P}$, and so on.

In general the containments do not hold in the other direction: For instance, $\mathrm{Heur}_{n^{-c}}\mathrm{P}$ contains undecidable problems but HeurP does not. However, the class $(\mathrm{NP}, \mathrm{PComp})$ as a whole admits heuristic schemes if and only if it admits heuristic algorithms, as formalized in the following proposition.

Proposition 3.5. If $(\mathrm{BH},\mathcal{U}^{\mathrm{BH}}) \in \mathrm{Avg}_{1/n}\mathbf{C}$ (respectively, $\mathrm{Heur}_{1/n}\mathbf{C}$), then $(\mathrm{NP}, \mathrm{PComp}) \subseteq \mathrm{Avg}\mathbf{C}$ (respectively, $\mathrm{Heur}\mathbf{C}$). Here, $\mathbf{C}$ is one of P, BPP, or ZPP.

Proof. For concreteness, let us show that if $(\mathrm{BH},\mathcal{U}^{\mathrm{BH}})$ is in $\mathrm{Avg}_{1/n}\mathrm{P}$, then $(\mathrm{NP}, \mathrm{PComp}) \in \mathrm{AvgP}$. By completeness of $(\mathrm{BH},\mathcal{U}^{\mathrm{BH}})$ with respect to distributional reductions, it is sufficient to show that $(\mathrm{BH},\mathcal{U}^{\mathrm{BH}}) \in \mathrm{AvgP}$.

Let A be an errorless heuristic algorithm for $(\mathrm{BH},\mathcal{U}^{\mathrm{BH}})$ with failure probability $1/n$. Using A, we construct an errorless heuristic scheme $A'(\cdot;\cdot)$. The idea is to use self-reducibility and padding in order to map short instances of BH into longer ones. Since the error probability of A decreases with instance length, the scheme A' can solve any desired fraction of instances by choosing a padding of appropriate length.

We claim that the following A' is an errorless heuristic scheme for $(\mathrm{BH},\mathcal{U}^{\mathrm{BH}})$: $A'((M,x,1^t);N,\delta) = A((M,x,1^{t+\lceil 1/\delta\rceil});N+\lceil 1/\delta\rceil)$, where N is the length of the instance $(M,x,1^t)$. (When the input is not of the proper form $(M,x,1^t)$, A' rejects it.) From the definition of the ensemble $\mathcal{U}^{\mathrm{BH}}$, we have that for all N,

$$U^{\mathrm{BH}}_{N+\lceil 1/\delta\rceil}(M,x,1^{t+\lceil 1/\delta\rceil}) = U^{\mathrm{BH}}_N(M,x,1^t).$$

On inputs from distribution $U^{\mathrm{BH}}_{N+\lceil 1/\delta\rceil}$, A outputs $\perp$ on at most a $1/(N+\lceil 1/\delta\rceil) < \delta$ fraction of instances, so it follows that A' outputs $\perp$ on at most a δ fraction of instances from U^{BH}_N. □

In fact, the error parameter $1/n$ in Proposition 3.5 can be replaced with $1/n^{\varepsilon}$ for any fixed $\varepsilon > 0$.

4

Decision Versus Search and One-Way Functions

In worst-case complexity, a search algorithm A for an NP-relation V is required to produce, on input x, a witness w of length poly($|x|$) such that V accepts $(x; w)$, whenever such a w exists. Abusing terminology, we sometimes call A a search algorithm for the NP-language L_V consisting of all x for which such a witness w exists. Thus, when we say "a search algorithm for L" we mean an algorithm that on input $x \in L$ outputs an NP-witness w that x is a member of L, with respect to an implicit NP-relation V such that $L = L_V$.

Designing search algorithms for languages in NP appears to be in general a harder task than designing decision algorithms. An efficient search algorithm for a language in NP immediately yields an efficient decision algorithm for the same language. The opposite, however, is not believed to be true in general (for instance, if one-way permutations exist, even ones that are hard to invert in the worst case). However, even though search algorithms may be more difficult to design than decision algorithms for specific problems, it is well known that search is no harder than decision for the class NP as a whole: If P = NP, then every language in NP has an efficient (worst-case) search algorithm.

In this section we revisit the question of decision versus search in the average-case setting: If all languages in distributional NP have good-on-average decision algorithms, do they also have good-on-average search algorithms? The answer was answered in the affirmative by Ben-David *et al.*, though for reasons more subtle than in the worst-case setting. Their argument yields search to decision connections even for interesting subclasses of distributional NP. For instance, if every language in NP is easy-on-average for decision algorithms with respect to the uniform distribution, then it is also easy-on-average for search algorithms with respect to the uniform distribution. We present their argument in Section 4.2.

From a cryptographic perspective, the most important distributional search problem in NP is the problem of inverting a candidate one-way function. By the argument of Ben-David *et al.*, if all problems in distributional NP are easy-on-average, then every candidate one-way function can be inverted on a random *output*. In Section 4.3 we will see that this conclusion holds even under the weaker assumption that every problem in NP is easy-on-average with respect to the uniform distribution. Thus, cryptographic one-way functions can exist only if there are problems in (NP,$\mathcal{U}$) that are hard-on-average for decision algorithms.

The search-to-decision reduction presented in this section yields *randomized* search algorithms for distributional NP. We begin by defining the types of search algorithms under consideration.

4.1 Search algorithms

By analogy with worst-case complexity, it is easiest to define search algorithms for NP whose running time is polynomial-on-average. For illustration, we present the definition for deterministic algorithms.

Definition 4.1. (average polynomial-time search) For an NP language L and ensemble of distributions $\mathcal{D}$, we say A is a *deterministic average polynomial-time search algorithm* for $(L,\mathcal{D})$, if for every n and every x in L and in the support of D_n, $A(x;n)$ outputs an L-witness for x and there exists a constant ε such that for every n, $\mathbf{E}_{x\sim D_n}[t_A(x;n)^\varepsilon] = O(n)$.

As in the case of decision algorithms, the existence of average polynomial-time search algorithms is equivalent to the existence of errorless heuristic search algorithms, which we define next. In the case of randomized algorithms, the adjective "errorless" refers to the random choice of an input from the language, and not to the choice of random coins by the algorithm. To make this distinction clear, we first define "errorless search" in the deterministic case, then extend the definition to the randomized case.

Definition 4.2. (deterministic errorless search) We say A is a *deterministic errorless search scheme* for $(L, \mathcal{D})$, where $L \in \text{NP}$, if there is a polynomial p such that

(1) For every $n, \delta > 0$, and every x in the support of D_n, $A(x; n, \delta)$ runs in time at most $p(n/\delta)$.
(2) For every $n, \delta > 0$, and every x in L and in the support of D_n, $A(x; n, \delta)$ outputs either an L-witness w for x or $\perp$.
(3) For every n and every $\delta > 0$, $\mathbf{Pr}_{x \sim D_n}[A(x; n, \delta) = \perp] \leq \delta$.

Observe that when $x \notin L$, the output of the algorithm can be arbitrary. If the algorithm outputs anything other than the special symbol $\perp$, this provides a certificate that x is not in L, as it can be efficiently checked that the output of the algorithm is not a witness for x.

In the case of randomized algorithms, we can distinguish different types of errors that the algorithm makes over its randomness. A "zero-error" randomized search algorithm is required to output, for all $x \in L$, either a witness for x or $\perp$ with probability one over its randomness. The type of search algorithm we consider here is allowed to make errors for certain choices of random coins; namely, even if $x \in L$, the search algorithm is allowed to output an incorrect witness with probability bounded away from one.

Definition 4.3. (randomized errorless search) We say A is a *randomized errorless search algorithm* for $(L, \mathcal{D})$, where $L \in \mathrm{NP}$, if there is a polynomial p such that

(1) For every $n, \delta > 0$, A runs in time $p(n/\delta)$ and outputs either a string w or the special symbol $\perp$.
(2) For every $n, \delta > 0$ and $x \in L$,

$$\mathbf{Pr}_A[A(x; n, \delta) \text{ outputs a witness for } x \text{ or } A(x; n, \delta) = \perp] > 1/2.$$

(3) For every n and $\delta > 0$,

$$\mathbf{Pr}_{x \sim D_n}\left[\mathbf{Pr}_A[A(x; n, \delta) = \perp] > 1/4\right] \leq \delta.$$

This definition is robust with respect to the choice of constants $\frac{1}{2}$ and $\frac{1}{4}$; it would remain equivalent if $\frac{1}{2}$ and $\frac{1}{4}$ were replaced by any two constants c and c', respectively, where $0 < c' < c < 1$. Using standard error reduction be repetition, the constants $\frac{1}{2}$ and $\frac{1}{4}$ can be amplified to $1 - \exp(-(n/\delta)^{O(1)})$ and $\exp(-(n/\delta)^{O(1)})$, respectively.

Finally, we define heuristic search algorithms: Such algorithms are allowed to output incorrect witnesses on a small fraction of inputs.

Definition 4.4. (randomized heuristic search) We say A is a *randomized heuristic search algorithm* for $(L, \mathcal{D})$, where $L \in \mathrm{NP}$, if for every n, on input x in the support of D_n and parameter $\delta > 0$, A runs in time polynomial in n and $1/\delta$, and

$$\mathbf{Pr}_{x \sim D_n}\left[x \in L \text{ and } \mathbf{Pr}_A[A(x; n, \delta) \text{ is not a witness for } x] > 1/4\right] \leq \delta.$$

4.2 Reducing search to decision

It is well known in worst-case complexity that the hardness of search and decision versions of NP-complete problems are equivalent. Namely, if any NP-complete problem has an efficient decision algorithm (on all instances), then not only does all of NP have efficient decision algorithms, but also all of NP have efficient search algorithms. The same question can be asked for distributional NP: If every decision problem in NP has good-on-average algorithms with respect to, say, the uniform

distribution, does every search problem in NP also have efficient algorithms with respect to the uniform distribution?

We show a result of Ben-David *et al.* that establishes the equivalence of search and decision algorithms for NP with the uniform distribution. We focus on the uniform distribution not only because it is the most natural distribution of instances, but also because the equivalence of search and decision complexities for the uniform distribution will be used to establish a much more general result in Section 5.1.

Let us recall the common argument used to establish the equivalence of NP-hardness for search and decision problems in the worst-case setting, and see why this argument fails to carry over directly to the average-case setting. Given a decision oracle for NP, and an instance x of an NP-language L, a search algorithm for x finds a witness by doing binary search for the lexicographically smallest w such that the oracle answers "yes" on the NP-query:

> (x, w): Is there an L-witness for x that is lexicographically at most w?

To see why this reduction is useless in the average-case setting with respect to the uniform distribution, fix the lexicographically smallest witness w_x for every $x \in L$, and suppose that the average-case decision oracle answers all queries correctly, except those (x, w) where the distance between w and w_x in the lexicographic order is small. Then the algorithm obtains only enough information from the oracle to recover the first few significant bits of w_x and cannot efficiently produce a witness for x.

To understand the idea of Ben-David *et al.*, let us first consider the special case when L is an NP language with *unique* witnesses. Given an input x, the reduction attempts to recover a witness for x by making oracle queries of the type

> (x, i): Does there exist a witness w for x such that the ith bit w_i of w is 1?

for every $i = 1, \ldots, m(|x|)$, where $m(n)$ is the length of a witness on inputs of length n. (Since $L \in \text{NP}$, we have that $m(n) = \text{poly}(n)$.) Given

a worst-case decision oracle for this NP language, the sequence of oracle answers on input $x \in L$ allows the search algorithm to recover all the bits of the unique witness w. In this setting, the reduction also works well on average: Given an average-case decision oracle that works on a $1 - \delta/m(n)$ fraction of inputs (x, i), where $|x| = n$ and $i \leq m(n)$, the search algorithm is able to recover witnesses (if they exist) on a $1 - \delta$ fraction of inputs $x \sim U_n$.

In general, witnesses need not be unique. However, using the isolating technique of Valiant and Vazirani [73], it is possible to (randomly) map instances of L to instances of another NP-language L' in such a way that (i) the distribution of each query is dominated by uniform; (ii) if x maps to x', then any witness that $x' \in L'$ is also a witness that $x \in L$, and (iii) If $x \in L$, then x maps to an instance $x' \in L'$ with a unique witness with non-negligible probability.

The language L' is defined as follows:

$$L' = \{(x, h, i, j) : \text{there exists an } L\text{-witness } w \text{ for } x \\ \text{such that } w_i = 1 \text{ and } h|_j(w) = 0^j\},$$

where i and j are numbers between 1 and $m(n)$, and h is a hash function mapping $\{0,1\}^{m(n)}$ to $\{0,1\}^{m(n)}$. The argument of Valiant and Vazirani guarantees that if j is the logarithm of the number of L-witnesses for x, there is a unique w satisfying $h|_j(w) = 0$ with constant probability over the choice of h. The reduction R, on input $x \sim U_n$, chooses a random hash function $h : \{0,1\}^{m(n)} \to \{0,1\}^{m(n)}$ and queries the average-case oracle for L' on instances (x, h, i, j), for all i, j between 1 and $m(n)$.

If, for any j, the sequence of answers to the queries (x, h, i, j) received from the oracle is an L-witness for x, the search algorithm for L outputs this witness. If no witness is found, a heuristic search algorithm outputs an arbitrary string. An errorless algorithm outputs the special symbol $\perp$ if this symbol was ever encountered as an answer to a query and an arbitrary string otherwise.

Theorem 4.5. (Ben-David *et al.*) If (NP,$\mathcal{U}$) $\subseteq$ AvgBPP (respectively, HeurBPP), then every problem in (NP,$\mathcal{U}$) has an errorless (respectively, heuristic) randomized search algorithm.

Observe that the search-to-decision reduction only applies to decision algorithms that succeed on most instances. For the argument to achieve non-trivial parameters, the fraction of instances on which the decision algorithm fails must be smaller than $1/m(n)^2$.

4.3 Average-case complexity and one-way functions

If every problem is easy-on-average for the uniform ensemble, can one-way functions exist? The above arguments show that in the case for one-way permutations, the answer is no. Given any efficiently constructible family of permutations $f_n : \{0,1\}^n \to \{0,1\}^n$ solving the search problem "Given y, find $f_n^{-1}(y)$" on most y chosen from the uniform ensemble gives the ability to invert $f_n(x)$ on a randomly chosen $x \sim U_n$.

In the general case, the answer is not immediately clear; to illustrate, consider the case of a function $f_n : \{0,1\}^n \to \{0,1\}^n$ whose image has density $2^{-n/2}$ in $\{0,1\}^n$ under the uniform distribution. An average-case inversion algorithm for f_n may fail to answer any queries that fall into the image of f_n, yet be efficient with respect to the uniform distribution by not failing on the other queries.

To rule out the existence of general one-way functions in this setting, it is sufficient (by Håstad *et al.* [38]) to show that no pseudorandom generators exist. We argue that this is the case in the errorless setting, that is under the assumption $(\mathrm{NP},\mathcal{U}) \subseteq \mathrm{AvgBPP}$. Given a candidate pseudorandom generator $G_n : \{0,1\}^{n-1} \to \{0,1\}^n$, consider the NP decision problem "Is y in the image set of $G_{|y|}$?" An errorless algorithm A for this problem must always answer "yes" or $\perp$ when the input is chosen according to $G_n(U_{n-1})$. On the other hand, $A(y;n,1/4)$ must answer "no" on at least a $\frac{1}{4}$ fraction of inputs $y \sim U_n$, since at least a $\frac{1}{2}$ fraction of such inputs is outside the image of G_n, and the algorithm is allowed to fail on no more than a $\frac{1}{4}$ fraction of inputs. Hence A distinguishes $G_n(U_{n-1})$ from the uniform distribution, so G_n is not a pseudorandom generator.

In the case of heuristic algorithms, this argument fails because there is no guarantee on the behavior of A on inputs that come from $G_n(U_{n-1})$. However, a different argument can be used to rule out one-way functions under this more restrictive assumption. Håstad et al.

show that if one-way functions exist, then a form of "almost one-way permutations" exists: There is a family of strongly one-way efficiently constructible functions $f_n : \{0,1\}^n \to \{0,1\}^n$ such that the image of f_n has non-negligible density in $\{0,1\}^n$, that is, $U_n(f_n(\{0,1\}^n)) = \sum_{x \in \text{Image}(f_n)} U_n(x) \geq n^{-O(1)}$. By choosing parameters appropriately, every such family of functions can be inverted on a large fraction of the image set $f_n(\{0,1\}^n)$. This gives an algorithm that inverts $f_n(x)$ on a non-neglibible fraction of *inputs* x and contradicts the assumption that f_n is strongly one way.

In Chapter 5, we give a different proof of this result that bypasses the analysis of Håstad *et al.* Summarizing, and using the equivalence of weakly and strongly one-way functions, we have the following:

Theorem 4.6. If $(\text{NP},\mathcal{U}) \subseteq \text{HeurBPP}$, then for every polynomial-time computable family of functions $f_n : \{0,1\}^n \to \{0,1\}^*$ there is a randomized algorithm $I(y;n,\delta)$ running in time polynomial in n and $1/\delta$ such that for every n and $\delta > 0$,

$$\mathbf{Pr}_{x \sim U_n}[I(f_n(x);n,\delta) \in f_n^{-1}(f_n(x))] \geq 1 - \delta.$$

5

Samplable Ensembles

The worst-case NP hardness of computational problems does not always reflect their perceived difficulty in practice. A possible explanation for this apparent disconnect is that even if a problem may be hard to solve in the worst case, hard instances of the problem are so difficult to generate that they are never encountered. This raises the intriguing possibility that an NP-hard problem, for instance SAT, does not have an efficient algorithm in the worst case, but generating a hard instance of SAT is in itself an infeasible problem. More precisely, for every sampler of presumably hard instances for SAT, there is an efficient algorithm that solves SAT on most of the instances generated by the sampler.

When the distribution of instances is known in advance, it makes sense to restrict attention to a fixed sampler and design algorithms that work well with respect to the output distribution of this sampler. This is a viewpoint commonly adopted in average-case algorithm design, where newer algorithms for problems such as kSAT are designed that work well on average for larger and larger classes of distributions on inputs. From a complexity theoretic perspective, on the other hand, one is more interested in the inherent limitations of average-case algorithms, and it

is natural to think of the sampler as chosen by an adversary that tries to generate the hardest possible instances of the problem.

How much computational power should such a sampler of "hard" instances be allowed? It does not make sense to give the sampler more computational power than the solver, since the solver must have at least sufficient time to parse the instance generated by the sampler. On the other hand, in practice the sampler will have access to the same computational resources as the solver, so if our notion of "efficient on average" solver is that of a polynomial-time algorithm, the sampler should also be allowed to perform arbitrary polynomial-time computations. This motivates the study of the distributional class $(\mathrm{NP}, \mathrm{PSAMP})$.

Even though instances drawn from a samplable ensemble may be harder than instances drawn from a computable (or from the uniform) ensemble for a specific problem in NP, it turns out this is not the case for the class NP as a whole: If uniformly distributed inputs are easy for every problem in NP, then so are inputs drawn from an arbitrary samplable ensemble.

5.1 The compressibility perspective

In Chapter 3 we showed that the distributional problem $(\mathrm{BH}, \mathcal{U}^{\mathrm{BH}})$ is complete for the class $(\mathrm{NP}, \mathrm{PCOMP})$. We did so by giving a reduction that maps instances of an arbitrary distributional problem $(L, \mathcal{D})$ in $(\mathrm{NP}, \mathrm{PCOMP})$ to instances of $(\mathrm{BH}, \mathcal{U}^{\mathrm{BH}})$.

Recall that the key idea of the proof was to find an efficiently computable mapping C with the following properties:

(1) The map C is injective, or equivalently, the encoding computed by C is uniquely decodable.
(2) When x is distributed according to $\mathcal{D}$, the output $C(x)$ is distributed "almost" uniformly. If we think of C as a compression procedure, it means that the rate of C is close to optimal.

In general it is not clear whether an encoding C with such properties exists for arbitrary samplable ensembles. Our approach will be to gradually relax these properties until they can be satisfied for all samplable ensembles $\mathcal{D}$.

To relax these properties, we look at randomized encodings. First, observe that randomness can be added to the encoding without affecting the correctness of the reduction: Suppose that C is a mapping such that when x is chosen according to the ensemble $\mathcal{D}$, the image $C(x)$ is distributed almost uniformly. Define a random mapping C' that, on input x, chooses a uniformly random string r of some fixed length and outputs the pair $(C(x), r)$. It is evident that if the mapping C satisfies conditions (1)–(3), then so does the mapping C'. We use $C'(x;r)$ to denote the output of C' on input x and randomness r; thus $C'(x;r) = (C(x), r)$.

The advantage of a randomized encoding is that et allows for a natural relaxation of condition (1): Instead of requiring that the mapping be injective, we can now consider encodings that are "almost injective" in the sense that given $C'(x;r)$, the encoding needs to be uniquely decodable only with high probability over r.

In fact, we will further weaken this requirement substantially, and only require that $C'(x;r)$ be uniquely decodable with non-negligible probability. Then a query made by the reduction is unlikely to be uniquely decodable, but by running the reduction several times we can expect that with high probability, at least one run of the reduction will yield a uniquely decodable query.

To summarize, we have the following situation: We are given a reduction that queries $(\mathrm{BH}, \mathcal{U}^{\mathrm{BH}})$ on several instances, and which expects to obtain the correct answer for at least one of these instances. We do not know which of the instances produced by the reduction is the good one, but since BH is an NP problem, instead of asking for a yes/no answer to the queries we can in fact ask for a witness that at least one of the queries produced by the reduction is a "yes" instance of BH. In fact, the search to decision reduction from Chapter 4 shows that obtaining a witness is no harder than obtaining a membership answer (for randomized reductions).

There is one important complication that we ignored in the last paragraph. Many of the queries produced by the reduction may not be uniquely decodable. Such queries may turn out to be "yes" instances of BH even if x was a "no" instance of L, thereby certifying that a query y is a "yes" instance of BH is not sufficient to conclude that $x \in L$.

Indeed, we will need to certify not only that $y \in \mathrm{BH}$, but also that y is uniquely decodable.

5.1.1 Reductions between search problems

We now formalize the properties of the reduction from the above discussion. Since the reduction needs to access witnesses for membership of its queries, we formalize it as a reduction between search problems. We only consider the case when one is reducing to a problem with respect to the uniform distribution, as this is our case of interest.

For two distributional problems $(L, \mathcal{D})$ and $(L', \mathcal{U})$ in $(\mathrm{NP}, \mathrm{PSAMP})$, a *randomized heuristic search reduction* from $(L, \mathcal{D})$ to $(L', \mathcal{U})$ is an algorithm R that takes an input x and a parameter n and runs in time polynomial in n, such that for every n and every x, there exists a set $V_x \subseteq \operatorname{Supp} R(x; n)$ (corresponding to the "uniquely decodable" queries) with the following properties:

(1) Disjointness: There is a polynomial p such that for every n, $V_x \subseteq \{0,1\}^{p(n)}$ and the sets V_x are pairwise disjoint.
(2) Density: There is a polynomial q_1 such that for every n and every x in the support of D_n,

$$\mathbf{Pr}_R[R(x; n) \in V_x] \geq 1/q_1(n).$$

(3) Uniformity: For every n and every x in the support of D_n, the distribution of queries $y \sim R(x; n)$ conditioned on $y \in V_x$ is uniform.
(4) Domination: There is a polynomial q_2 such that for every n and every x,

$$D_n(x) \leq q_2(n) \cdot U_{p(n)}(V_x).$$

(5) Certifiability: There exists a polynomial-time algorithm Q such that for every n, if $x \in L$ and $y \in V_x$, then for every L'-witness w for y, $Q(w)$ is an L-witness for x.

A randomized search reduction is weaker than a reduction between decision problems in that it is only guaranteed to work with small probability, and only on "yes" instances. However, if we are given a

randomized search algorithm for L', it gives a randomized search algorithm for L as well, since et allows us to recover witnesses for L from witnesses for L'. If we run the reduction several times, the probability we hit a witness for L' becomes exponentially close to one, so the search algorithm for L can be made to work with very high probability on all instances.

Claim 5.1. If there is a randomized search reduction from $(L,\mathcal{D})$ to $(L',\mathcal{U})$, and $(L',\mathcal{U})$ has a randomized heuristic search scheme, then $(L,\mathcal{D})$ has a randomized heuristic search scheme.

Proof. Let A' be a randomized heuristic search scheme for $(L',\mathcal{U})$. The search scheme A for $(L,\mathcal{D})$ will run the reduction N times, producing N search queries for A'. For each witness w_i returned by A', A will check whether w_i yields a witness for L.

Specifically, on input x and parameters n and δ, A does the following:

(1) Run $R(x;n)$ independently $N = 16q_1(n)$ times, producing queries $y_1,\dots,y_N$.
(2) Compute $w_i = A'(y_i;p(n),\delta/2q_2(n))$ for every i.
(3) If, for some i, $Q(w_i)$ is an L-witness for x, output $Q(w_i)$ (and otherwise output an arbitrary string).

Assume $x \in L$, and denote by F the set of all y on which $A'(y;\cdot)$ behaves incorrectly. Specifically, let F be the set of all y such that $y \in L'$ but $A'(y;p(n),\delta/2q_2(n))$ fails to return a witness of y with probability $\frac{1}{4}$ or more. Since A' is a heuristic scheme for L', we have that $U_{p(n)}(F) \le \delta/2q_2(n)$.

Let B be the set of all $x \in L \cap \operatorname{Supp} D_n$ for which a large portion of the uniquely decodable queries V_x are "bad" for A' in the sense that they fall inside F. Specifically, define B as the set of all x such that

$$U_{p(n)}(V_x \cap F) \ge U_{p(n)}(V_x)/2.$$

The set B cannot have much weight according to D_n, since every $x \in B$ is "responsible" for many bad queries in $V_x \cap F$, and if there

were many such queries then F would be large. In particular,

$$\begin{aligned} D_n(B) &= \sum_{x \in B} D_n(x) \\ &\leq \sum_{x \in B} q_2(n) U_{p(n)}(V_x) \qquad \text{(by domination)} \\ &\leq \sum_{x \in B} 2q_2(n) U_{p(n)}(V_x \cap F) \\ &\leq 2q_2(n) U_{p(n)}(F) \leq \delta \qquad \text{(by disjointness).} \end{aligned}$$

Now fix $x \notin B$, and consider one of the queries y_i generated by A in step (1). We have that

$$\begin{aligned} &\mathbf{Pr}[Q(w_i) \text{ is an } L\text{-witness for } x] \\ &\quad \geq \mathbf{Pr}[y_i \in V_x \text{ and } w_i \text{ is an } L'\text{-witness for } y_i] \quad \text{(by certifiability)} \\ &\quad \geq \mathbf{Pr}[y_i \in V_x - F \text{ and } w_i \text{ is an } L'\text{-witness for } y_i] \\ &\quad = \mathbf{Pr}[y_i \in V_x] \cdot \mathbf{Pr}[y_i \in V_x - F \mid y_i \in V_x] \\ &\qquad \times \mathbf{Pr}[w_i \text{ is an } L'\text{-witness for } y_i \mid y_i \in V_x - F] \\ &\quad \geq \frac{1}{q_1(n)} \cdot \frac{1}{2} \cdot \frac{1}{4} = \frac{1}{8q_1(n)}, \end{aligned}$$

by density, uniformity, and the definition of F. By the choice of N, it follows that at least one of $Q(w_1), \ldots, Q(w_N)$ is an L-witness for x with probability $\frac{1}{2}$. □

This claim shows that randomized search reductions can be used to prove completeness results for HeurBPP. However, the proof of the claim does not extend to the class AvgBPP, the reason being that the domination condition is too weak. For heuristic algorithms, this condition guarantees that the algorithm A' for $(L', \mathcal{U})$ will provide witnesses to most of the "yes" instances of $(L, \mathcal{D})$. The "evidence" that an instance of $(L, \mathcal{D})$ is a "no" instance is that no such witness is found.

In the case of errorless algorithms, however, we need to certify "no" instances of $(L, \mathcal{D})$. It is reasonable to attempt the following: First, run the reduction several times to estimate the fraction of queries that A' answers by $\perp$. If this fraction turns out too large, this is evidence that A' is unable to provide witnesses reliably for this instance, so

we answer $\perp$. Otherwise, we look for a witness and answer accordingly. Unfortunately, the definition is insufficient to guarantee that $\perp$ will not be answered too often, since it may be that the distribution of queries is skewed in such a way that, whenever a query for x falls outside V_x, the answer to this query is very likely to be $\perp$.

5.1.2 Compressing arbitrary samplable distributions

Let S be a polynomial-time sampler that on input n runs in time $m(n)$, where m is some polynomial and D_n denotes the distribution of the random variable $S(n)$. As for computable distributions, our goal is to extract a sufficient amount of randomness from $S(n)$ so that the output of the extraction procedure is dominated by the uniform distribution.

To describe the approach, it is convenient to begin by considering the problem for certain restricted classes of distributions D_n, then gradually remove the restrictions until all samplable distributions are encompassed.

We begin by considering the case of flat distributions: We say that D_n is k_n-*flat* if for each x in the support of D_n, $D_n(x) = 2^{-k_n}$. Flat distributions are convenient to consider because their randomness can be extracted via the Leftover Hash Lemma: In particular, when x is chosen from a k_n-flat distribution and h is a random hash function from $\{0,1\}^{<m(n)}$ into $\{0,1\}^{k_n+7}$, the output of the mapping $C_n(x;h) = (h, h(x))$ is dominated by the uniform distribution. It is not difficult to check that C_n satisfies the properties of randomized heuristic search reductions: The "uniquely decodable" strings V_x are those pairs (h, y) for which $h^{-1}(y) = \{x\}$. By the choice of parameters, for every x in the support of D_n, $(h, h(x)) \in V_x$ for all but a small fraction of possible h, giving both density and domination. (Uniformity and certifiability are trivial.)

Now we consider a small but important generalization of flat distributions: Instead of requiring that all samples in the support of D_n have the same probability, we allow their probabilities to vary, but require that these probabilities be efficiently computable in the following sense: There is an algorithm that on input x and parameter n runs in time polynomial in n and computes the approximate entropy of x, which is the value

$$k_n(x) = \lfloor -\log_2 D_n(x) \rfloor = m(n) - \lceil \log_2 \#\{r : S(n;r) = x\} \rceil.$$

Notice that $k_n(x)$ is an integer between 0 and $m(n)$. This scenario subsumes the previous one, where $k_n(x)$ was the same for all x in the support of D_n. The reasoning for flat distributions extends to this scenario, as long as we tailor the length of the output of the hash function to depend on the entropy $k(x)$. Namely, the mapping $C_n(x;h) = (h, h|_{k_n(x)+7}(x))$, where h is a function mapping $\{0,1\}^{<m(n)}$ to $\{0,1\}^{m(n)+7}$ satisfies the properties of randomized heuristic search reductions.

For arbitrary S, $k_n(x)$ could be difficult to compute and it is not clear if the approach of compressing samples via hashing can be extended. One idea is for the reduction to attempt all possible values for $k_n(x)$, and declare V_x to be the subset of encodings for which the guess was correct. However, it is now possible that strings of higher entropy (lower probability) than x become possible decodings of $(h, h(x))$: There may be many such strings, and it is likely that some of them collide with x under h.

The solution is to append the encoding $C_n(x)$ of x with a "certificate" that the entropy of x is not too high, namely that $k_n(x) \leq k$. This roughly amounts to certifying that the size of the set $\{r : S(n;r) = x\}$ is at least $2^{m(n)-k_n}$. The certificate of this statement will be randomized: We ask to see a string r such that $S(r) = x$ and $g(r) = 0$ for a random hash function g that is approximately 2^{k_n}-to-one. Such a certificate is only approximately correct, but this is sufficient to guarantee that with constant probability, for a random h, $h(x)$ has a unique pre-image for h mapping $\{0,1\}^{<m(n)}$ to $\{0,1\}^{k_n+7}$.

5.1.3 The construction

Putting everything together, the encoding for x chosen from distribution D_n is

$$C_n(x;h,g,k) = (h(x), h, g, k),$$

where k is a number between 0 and $m(n)$, h is a hash function mapping $\{0,1\}^{<m(n)}$ to $\{0,1\}^{k+7}$, and g is a hash function mapping $\{0,1\}^{m(n)}$ to $\{0,1\}^{m(n)-k-4}$. (In reality, h maps to $\{0,1\}^{m(n)+7}$ and g maps to $\{0,1\}^{m(n)-4}$ and we use the truncated versions $h|_{k+7}$ and $g|_{m(n)-k-4}$,

but for simplicity of notation we will not make this distinction.) Let $p(n)$ denote the output length of C_n.

The "uniquely decodable" encodings are defined as follows:

> V_x is the set of all (y, h, g, k) such that $k = k_n(x)$, $h(x) = y$, and
>
> (1) There is an r such that $S(n;r) = x$ and $g(r) = 0$.
>
> (2) If $h(S(n;r)) = y$ and $g(r) = 0$, then $S(n;r) = x$.

The reduction R maps instance $(x;n)$ to instance $(h(x), h, g, k)$ of the following NP-language L':

> $(y, h, g, k) \in L'$ if there exists an r of length $< m(n)$ such that $S(n;r) \in L$ and $h(S(n;r)) = y$ and $g(r) = 0^{m(n)-k-4}$.

Observe that a certificate that $(y, h, g, k) \in L'$ in particular contains a certificate that $S(n;r) \in L$, so under appropriate conditions witnesses for membership in L can be extracted from the corresponding witnesses for L'.

Theorem 5.2. (Impagliazzo and Levin) $(L, \mathcal{D})$ reduces to $(L', \mathcal{U})$ via a randomized heuristic search reduction.

Combining this result with the completeness of $(\mathrm{BH}, \mathcal{U}^{\mathrm{BH}})$ for problems in $(\mathrm{NP}, \mathcal{U})$, which follows from Cook's reduction (or as a special case of Theorem 3.3), and also using the search-to-decision equivalence of Theorem 4.5, we obtain the following corollary.

Corollary 5.3. If $(\mathrm{BH}, \mathcal{U}^{\mathrm{BH}}) \in \mathrm{HeurBPP}$, then $(\mathrm{NP}, \mathrm{PSAMP}) \subseteq \mathrm{HeurBPP}$.

Proof of Theorem 5.2. We show that the reduction R satisfies the five conditions for randomized heuristic search reductions. Let us fix n. Disjointness, uniformity, and certifiability follow from the definitions, so we focus on density and closeness.

Let $k_n(x) = \lfloor -\log_2 D_n(x) \rfloor = m(n) - \lceil \log_2 |\{r : S(n;r) = x\}| \rceil$. Let $p(n)$ denote the length of the output of the reduction when x is chosen from D_n.

Density: We show that $\mathbf{Pr}_{h,g}[(h(x), h, g, k) \in V_x]$ is lower bounded by a constant conditioned on $k = k_n(x)$. Since $k = k_n(x)$ with probability at least $1/m(n)$, it will follow that

$$\mathbf{Pr}_R[(h(x), h, g, k) \in V_x] = \Omega(1/m(n)).$$

We first show that with probability $\frac{7}{8}$, there exists an r such that $S(n;r) = x$ and $g(r) = 0$. Observe that the number of rs satisfying $S(n;r) = x$ is at least $2^{m(n)-k-1}$. Since the range of g is $\{0,1\}^{m(n)-k-4}$, in expectation there are at least eight rs such that $S(n;r) = x$ and $g(r) = 0$. By the pairwise independence of g, at least one r satisfies these conditions with probability $\frac{7}{8}$.

We now show that there are at most $\frac{1}{8}$ fraction of pairs h, g such that $h(S(n;r)) = y$ and $g(r) = 0$ for some r with $S(n;r) \neq x$. Indeed,

$$\begin{aligned}
&\mathbf{Pr}_{h,g}[\exists r : S(n;r) \neq x \text{ and } h(S(n;r)) = h(x) \text{ and } g(r) = 0] \\
&\qquad \leq \sum_{r : S(n;r) \neq x} \mathbf{Pr}_h[h(S(n;r)) = h(x)]\, \mathbf{Pr}_g[g(r) = 0] \\
&\qquad \leq \sum_{r \in \{0,1\}^{<m(n)}} 2^{-k-7} 2^{-m(n)+k+4} = 1/8.
\end{aligned}$$

It follows that each of the conditions (1) and (2) in the definition of V_x is satisfied with probability $\frac{7}{8}$ separately, so that

$$\mathbf{Pr}_{h,g}[(h(x), h, g, k) \in V_x \mid k = k_n(x)] \geq 3/4.$$

Domination: Observe that for a given n, a random instance of $U^{\mathrm{BH}}_{p(n)}$ is a 4-tuple of the correct form (y, h, g, k) with probability at least $1/\mathrm{poly}(p(n))$. Therefore,

$$\begin{aligned}
U^{\mathrm{BH}}_{p(n)}(V_x) &= \mathbf{Pr}_{y,g,h,k}[(y, h, g, k) \in V_x] \cdot 1/\mathrm{poly}(p(n)) \\
&\geq \mathbf{Pr}_{h,g}[(h(x), h, g, k) \in V_x \mid k = k_n(x)] \\
&\quad \mathbf{Pr}_y[y = h(x) \mid k = k_n(x)]\, \mathbf{Pr}_k[k = k_n(x)] \cdot 1/\mathrm{poly}(p(n)) \\
&\geq 3/4 \cdot 2^{-k_n(x)-7} \cdot 1/(m(n)\mathrm{poly}(p(n))) \\
&= \Omega(D_n(x)/m(n)\mathrm{poly}(p(n))).
\end{aligned}$$

□

An important example of a problem in (NP, PSAMP) is the problem of inverting a supposed one-way function $f_n : \{0,1\}^n \to \{0,1\}^*$: The question of finding an inverse $f_n^{-1}(y)$ is an NP question, and the distribution ensemble on which the function ought to be inverted is $\{f_n(U_n)\}$. Therefore, if (BH,$\mathcal{U}^{\mathrm{BH}}$) has a heuristic scheme, then no one-way functions exist.

5.2 The invertibility perspective

In this section we present a different proof that (NP, PSAMP) is no harder on average than (NP,$\mathcal{U}$) for randomized algorithms. This proof works for heuristic as well as errorless algorithms.

Ignoring efficiency considerations for the moment, given an NP language L and a polynomial-time sampler S, the distributional problem "Compute f on input x", where $x \sim S(n; U_{m(n)})$, can be solved first by sampling a random $r \sim U_{m(n)}$ conditioned on $S(n;r) = x$, and then by solving the distributional problem "Compute $f(S(r))$ on input r." Observe that given an algorithm that solves the latter problem well on average with respect to the uniform ensemble yields an algorithm for the original problem with respect to the ensemble $S(n; U_{m(n)})$.

The difficulty, of course, is in efficiently carrying out the step of sampling a random r conditioned on $S(n;r) = x$. In a general setting this does not seem possible, as $S(n;r)$ may be a one-way function of r, in which case finding any, let alone a random preimage of x, is an impossible task.

However, if all of (NP,$\mathcal{U}$) has efficient on average algorithms, by Theorem 4.6 there are no one-way functions. Impagliazzo and Luby [43] show that if there are no one-way functions then there are no *distributionally* one-way functions: Given any efficiently computable family of functions $f_n : \{0,1\}^n \to \{0,1\}^*$, for most y it is possible to efficiently sample an x such that $f_n(x) = y$ and the distribution of x conditioned on $f_n(x) = y$ is close to uniform. More precisely, there exists a (randomized) algorithm I running in time polynomial in n and $1/\delta$ such that the statistical distance between the distributions $(x, f_n(x))$ and $(I(f_n(x); n, \delta), f_n(x))$ is at most δ. In particular, given an input $x \sim$

$S(n;U_{m(n)})$, it is possible to sample an almost uniform r such that $S(n;r) = x$.

Theorem 5.4. (Impagliazzo and Levin) If $(\mathrm{NP},\mathcal{U}) \subseteq \mathrm{AvgZPP}$ (respectively, HeurBPP), then $(\mathrm{NP}, \mathrm{PSAMP}) \subseteq \mathrm{AvgZPP}$ (respectively, HeurBPP).

Proof. Consider an arbitrary problem $(L,\mathcal{D}) \in (\mathrm{NP}, \mathrm{PSAMP})$. Let S be the polynomial-time sampler for $\mathcal{D}$. Assume without loss of generality that on input n, S uses exactly $m(n)$ random bits and that m is an injective function. Under the assumption of the theorem, by Theorem 4.6 and the result of Impagliazzo and Luby, there is an algorithm I running in time polynomial in n and $1/\delta$ and such that for every n, the statistical distance between the distributions

$$\{(r, S(r)) : r \in \{0,1\}^{m(n)}\} \text{ and } \{(I(S(r)), S(r)) : r \in \{0,1\}^{m(n)}\} \quad (5.1)$$

is at most $\delta/3$. (For simplicity of notation, we omit the parameters n and δ in parts of the proof.) Let A be a heuristic scheme for the distributional problem $(L \circ S,\mathcal{U})$, where $L \circ S$ is the NP language $\{r : S(r) \text{ is a yes instance of } L\}$.

We show that the algorithm

$$B(x;n,\delta) = A(I(x);m(n),\delta/3)$$

is a heuristic scheme for $(L,\mathcal{D})$. Observe that if A is errorless then B is also errorless (since I can be made errorless by checking that S maps its input to its output, and outputing $\perp$ if this is not the case). Now, it is sufficient to show that

$$\mathbf{Pr}_{x\sim S(n;U_{m(n)})}[B(x) = L(x)] = \mathbf{Pr}_{r\sim U_{m(n)}}[B(S(r)) = L(S(r))] \geq 1 - \delta.$$

We relate the probability of the event $B(S(r)) = L(S(r))$ to the probability of the event $A(r) = L(S(r))$. By indistinguishability (5.1), for any event E, the probabilities of $E(r)$ and $E(I(S(r)))$ when $r \sim U_{m(n)}$ can differ by at most $\delta/3$, so in particular

$$\begin{aligned}
&\mathbf{Pr}_{r\sim U_{m(n)}}[A(r) = L(S(r))] \\
&\quad \leq \mathbf{Pr}_{r\sim U_{m(n)}}[A(I(S(r))) = L(S(I(S(r))))] + \delta/3 \\
&\quad = \mathbf{Pr}_{r\sim U_{m(n)}}[B(S(r)) = L(S(I(S(r))))] + \delta/3.
\end{aligned}$$

Applying indistinguishability (5.1) again, the distributions $(S(r), S(r))$ and $(S(I(S(r))), S(r))$ are $\delta/3$ statistically close, so in particular $\mathbf{Pr}_r[S(r) \neq S(I(S(r)))] < \delta/3$ and

$$\begin{aligned}
&\mathbf{Pr}_{r\sim U_{m(n)}}[B(S(r)) = L(S(I(S(r))))] \\
&\quad \leq \mathbf{Pr}_{r\sim U_{m(n)}}[B(S(r)) = L(S(I(S(r)))) \text{ and } S(r) = S(I(S(r)))] \\
&\qquad + \mathbf{Pr}_{r\sim U_{m(n)}}[S(r) \neq S(I(S(r)))] \\
&\quad \leq \mathbf{Pr}_{r\sim U_{m(n)}}[B(S(r)) = L(S(r))] + \delta/3.
\end{aligned}$$

Putting the last two equations together, we obtain

$$\begin{aligned}
&\mathbf{Pr}_{r\sim U_{m(n)}}[B(S(r)) = L(S(r))] \\
&\qquad \geq \mathbf{Pr}_{r\sim U_{m(n)}}[A(r) = L(S(r))] - 2\delta/3 \geq 1 - \delta.
\end{aligned}$$

□

Notice that the assumption that (NP,$\mathcal{U}$) has good on average algorithms was used twice in the proof: Once to invert the sampler S and once to solve $L \circ S$ on the uniform distribution. In other words, given an average-case oracle for (BH,$\mathcal{U}^{\mathrm{BH}}$), to obtain an algorithm for a problem in $(\mathrm{NP}, \mathrm{PSAMP})$ one needs to place two rounds of queries to the oracle. The first round of queries is used to obtain a pre-image r of x under S, and the second round (in fact, a single query) is used to solve $L \circ S$ on input r. In contrast, Theorem 5.2 solves problems in $(\mathrm{NP}, \mathrm{PSAMP})$ using a single round of oracle queries.

6

Hardness Amplification

Generally speaking, the goal of *hardness amplification* is to start from a problem that is known (or assumed) to be hard on average in a weak sense (i.e., every efficient algorithm has a noticeable probability of making a mistake on a random input) and to define a related new problem that is hard on average in the strongest possible sense (i.e., no efficient algorithm can solve the problem noticeably better than by guessing a solution at random).

6.1 Yao's XOR Lemma

For decision problems, Yao's XOR Lemma [76] is a very powerful result on amplification of hardnes. In the XOR Lemma, we start from a Boolean function $f : \{0,1\}^n \to \{0,1\}$ and define a new function $f^{\oplus k}(x_1, \ldots, x_k) := f(x_1) \oplus \cdots \oplus f(x_k)$, and the lemma says that if every circuit of size $\leq S$ makes at least a δ fraction of errors in computing $f(x)$ for a random x, then every circuit of size $\leq S \cdot \mathrm{poly}(\delta\varepsilon/k)$ makes at least a $1/2 - \varepsilon$ fraction of errors in computing $f^{\oplus k}$, where ε is roughly $\Omega((1-\delta)^k)$.

Various proofs of the XOR Lemma are known [16, 34, 41, 44, 54]. In this section we describe Impagliazzo's proof [41] because it is based

on a tool, Impagliazzo's "hard core distribution" theorem, that will be very useful later.

For simplicity, we will restrict ourselves to results in the non-uniform (circuit complexity) setting. The following definition will be useful.

Definition 6.1. We say that a Boolean function $f : \{0,1\}^n \to \{0,1\}$ is (S,δ)-hard with respect to a distribution D if, for every circuit C of size $\leq S$, we have

$$\mathbf{Pr}_{x\sim D}[f(x) \neq C(x)] > \delta.$$

To relate this definition to our previous definitions, observe that $(L,\{D_n\}) \in \mathrm{Heur}_{\delta(n)}\mathrm{SIZE}(S(n))$ if and only if, for every n, L_n is not $(S(n),\delta(n))$-hard with respect to D_n, where $L_n : \{0,1\}^n \to \{0,1\}$ is the characteristic function of the set $L \cap \{0,1\}^n$.

Impagliazzo [41] proves that, if a Boolean function is "mildly" hard on average with respect to the uniform distribution, then there is a large set of inputs such that the function is "very" hard on average on inputs coming from that set.

Lemma 6.2. (Impagliazzo) Let $f : \{0,1\}^n \to \{0,1\}$ be a (S,δ)-hard function with respect to the uniform distribution. Then, for every ε, there is a set $H \subseteq \{0,1\}^n$ of size $\delta 2^n$ such that f is $(S \cdot \mathrm{poly}(\varepsilon,\delta), \frac{1}{2} - \varepsilon)$-hard with respect to the uniform distribution over H.

We can now present Impagliazzo's proof of the XOR Lemma.

Theorem 6.3. (XOR Lemma, Impagliazzo's version) Let $f : \{0,1\}^n \to \{0,1\}$ be (S,δ)-hard with respect to the uniform distribution, let k be an integer, and define $g : \{0,1\}^{nk} \to \{0,1\}$ as

$$g(x_1,\ldots,x_k) := f(x_1) \oplus \cdots \oplus f(x_k)\ .$$

Then, for every $\varepsilon > 0$, g is $(S \cdot \mathrm{poly}(\varepsilon,\delta), \frac{1}{2} - \varepsilon - (1-\delta)^k)$-hard with respect to the uniform distribution.

Let H be a set as in Lemma 6.2. The main idea in the proof is that if we are a small circuit, then our chances of computing $f(x)$ for

$x \sim H$ are about the same as our chances of guessing the value of a random coin flip. Now, we are given $x_1, \ldots, x_k$ and we need to compute $f(x_1) \oplus \cdots \oplus f(x_k)$; if some x_j is in H, then, intuitively, our chances of correctly doing the computation are about the same as our chances of computing $f(x_1) \oplus \cdots \oplus f(x_{j-1}) \oplus b \oplus f(x_{j+1}) \cdots \oplus f(x_k)$, where b is a random bit. A random bit XOR-ed with other independent values is also a random bit, and so, in that case, we will be correct only with probability $\frac{1}{2}$. So our probability of being correct is at most $\frac{1}{2}$ plus $(1-\delta)^k$ (the probability that none of the x_j is in H) plus ε (to account for the difference between our ability to guess a random bit and our ability to compute $f(x)$ for $x \sim H$).

Even though this proof sketch may look completely unsound, it leads to a surprisingly simple formal proof, that we present below.

Proof of Theorem 6.3. Apply Lemma 6.2, and let H be the set of size $\delta 2^n$ such that f is $(S \cdot \mathrm{poly}(\varepsilon, \delta), \frac{1}{2} - \varepsilon)$-hard with respect to the uniform distribution over H.

Let C be a circuit of size S' such that

$$\mathbf{Pr}[C(x_1, \ldots, x_k) = f(x_1) \oplus \cdots \oplus f(x_k)] > \frac{1}{2} + (1-\delta)^k + \varepsilon.$$

Let D be the uniform distribution over k-tuples $(x_1, \ldots, x_k) \in (\{0,1\}^n)^k$ conditioned on at least one x_j being an element of H. By conditioning on the event that some $x_j \in H$, we obtain

$$\mathbf{Pr}_{(x_1,\ldots,x_k)\sim D}[C(x_1, \ldots, x_k) = f(x_1) \oplus \cdots \oplus f(x_k)] > \frac{1}{2} + \varepsilon.$$

We can see the process of picking a k-tuple $(x_1, \ldots, x_k) \sim D$ as first picking a non-empty subset $S \subseteq [k]$ with an appropriate distribution, then, for each $j \in S$, picking x_j uniformly from H, and, for each $j \notin S$, picking x_j uniformly from $\{0,1\}^n - H$, so the above expression can be rewritten as

$$\mathbf{E}_{S \neq \varnothing}\left[\mathbf{Pr}_{x_1,\ldots,x_k}[C(x_1, \ldots, x_k) = f(x_1) \oplus \cdots \oplus f(x_k)]\right] > \frac{1}{2} + \varepsilon.$$

Fix the set S that maximizes the outside expectation, and let i be the first element of S. Then we have

$$\mathbf{Pr}_{x_1,\ldots,x_k}[C(x_1, \ldots, x_k) = f(x_1) \oplus \cdots \oplus f(x_k)] > \frac{1}{2} + \varepsilon$$

or equivalently

$$\mathbf{E}_{x_1,\ldots,x_{i-1},x_{i+1},\ldots,x_n}\left[\mathbf{Pr}_{x_i\sim H}[C(x_1,\ldots,x_k) = f(x_1)\oplus\cdots\oplus f(x_k)]\right] > \frac{1}{2}+\varepsilon$$

Let a_j for $j \neq i$ be the assignment for x_j that maximizes the above expectation. Then we have

$$\mathbf{Pr}_{x_i\sim H}[C(a_1,\ldots,a_{i-1},x_i,a_{i+1},\ldots,a_k) = f(a_1)\oplus\cdots\oplus f(a_{i-1})\oplus f(x_i)\oplus f(a_{i+1})\oplus\cdots\oplus f(a_k)] > \frac{1}{2}+\varepsilon,$$

which we can rearrange as

$$\mathbf{Pr}_{x\sim H}[C(a_1,\ldots,a_{i-1},x_i,a_{i+1},\ldots,a_k) \oplus f(a_1)\oplus\cdots\oplus f(a_{i-1})\oplus f(a_{i+1})\oplus\cdots\oplus f(a_k) = f(x)] > \frac{1}{2}+\varepsilon.$$

Note that the left-hand side expression above can be computed by a circuit of size at most $S'+1$, showing that f is not $(S'+1, \frac{1}{2}-\varepsilon)$-hard with respect to the uniform distribution over H. We can choose $S' = S\cdot\mathrm{poly}(\varepsilon,\delta)$ in a way that contradicts our assumption about f being (S,δ)-hard with respect to U_n, and so we conclude that g is indeed $(S\cdot\mathrm{poly}(\varepsilon,\delta), \frac{1}{2}-\varepsilon-(1-\delta)^k)$-hard with respect to the uniform distribution. □

6.2 O'Donnell's approach

The XOR Lemma does not allow us to prove results of the form "if there is a midly hard-on-average distributional problem in NP with respect to the uniform distribution, then there is a very hard-on-average distributional problem in NP with respect to the uniform distribution." The difficulty is that if L is (the characteristic function of) a problem in NP, then, given x, y, it is not clear that the problem of computing $L(x)\oplus L(y)$ is still in NP. Indeed, if L is NP-complete, then computing $L(x)\oplus L(y)$ is not in NP unless NP = *co*NP.

We note, however, that if $g:\{0,1\}^k\to\{0,1\}$ is a *monotone* function, and L is in NP, then computing $g(L(x_1),\ldots,L(x_k))$ given $(x_1,\ldots,x_k)$ is a problem in NP. We may then ask whether there are monotone functions g such that, if L is mildly hard on average, then computing $g(L(x_1),\ldots,L(x_k))$ is very hard on average.

To address this question, we return to the informal proof of the XOR Lemma outlined in the previous section. Let $f : \{0,1\}^n \to \{0,1\}$ be a (S,δ)-hard function, and let H be a set as in Impagliazzo's Lemma. Define the probabilistic function F such that $F(x) = f(x)$ for $x \notin H$ and $F(x)$ is a random bit for $x \in H$. Our informal proof of the XOR Lemma was that, for a small circuit, computing $F(x_1) \oplus \cdots \oplus F(x_k)$ given $(x_1, \ldots, x_k)$ is about as hard as computing $f(x_1) \oplus \cdots f(x_k)$ given $(x_1, \ldots, x_k)$; no algorithm, however, can solve the former problem with probability larger than $\frac{1}{2} + (1-\delta)^k$, for information-theoretic reasons, and so this is also an approximate upper bound to the probability that a small circuit correctly solves the latter problem.

O'Donnell [61] shows that there are monotone functions g such that computing $g(F(x_1), \ldots, F(x_k))$ given $(x_1, \ldots, x_k)$ cannot be done with probability larger than $1/2 + \varepsilon$, provided k is at least $\mathrm{poly}(1/\varepsilon, 1/\delta)$, and a similar upper bound holds for the probability that a small circuit can compute $g(f(x_1), \ldots, f(x_k))$ given $(x_1, \ldots, x_k)$.

Let us start with a formalization of the information-theoretic result. For a function $f : \{0,1\}^n \to \{0,1\}$ and a set $H \subseteq \{0,1\}^n$, we denote by F_H a random variable distributed over functions $\{0,1\}^n \to \{0,1\}$, defined so that $F_H(x)$ is a random bit for $x \in H$ and $F_H(x) = f(x)$ for $x \notin H$. We say that a Boolean function is *balanced* if $\mathbf{Pr}[f(U_n) = 1] = \frac{1}{2}$.

Lemma 6.4. (O'Donnell) For every $\varepsilon > 0$, $\delta > 0$ there is a $k = \mathrm{poly}(1/\varepsilon, 1/\delta)$ and a monotone function $g : \{0,1\}^k \to \{0,1\}$, computable by a circuit of size $O(k)$, such that for every balanced function $f : \{0,1\}^n \to \{0,1\}$, every subset $H \subseteq \{0,1\}^n$ of size $\delta 2^n$ and every function $A : \{0,1\}^{kn} \to \{0,1\}$ we have

$$\mathbf{Pr}_{x_1,\ldots,x_k}[A(x_1, \ldots, x_k) = g(F_H(x_1), \ldots, F_H(x_k))] \leq \frac{1}{2} + \varepsilon,$$

where different occurrences of F_H in the above expression are sampled independendently.

The proof of the lemma is not easy, and we refer the reader to [61] for more details. Let us see how to use the lemma for the sake of hardness amplification. We need to formalize the notion of $g(F_H(x_1), \ldots, F_H(x_k))$

and $g(f(x_1),\ldots,f(x_k))$ being similarly hard to compute for a small circuit. Specifically, we prove the following result.

Lemma 6.5. Let $f:\{0,1\}^n \to \{0,1\}$ be a (S,δ)-hard function. Then, for every $\alpha > 0$, there is a set H of size $\delta 2^n$ such that for every k, and every function $g:\{0,1\}^k \to \{0,1\}$ computable by a circuit of size at most s, and for every circuit A of size at most $S \cdot \mathrm{poly}(\alpha,\delta) - s$, we have

$$\begin{aligned}\mathbf{Pr}[A(x_1,\ldots,x_k) &= g(f(x_1),\ldots,f(x_k))] \\ &\leq \mathbf{Pr}[A(x_1,\ldots,x_k) = g(F_H(x_1),\ldots,F_H(x_k))] + k\cdot\alpha\delta.\end{aligned}$$

In order to sketch the proof of Lemma 6.5, we first need to introduce the notion of *computational indistinguishability*. We say that two distributions X, Y ranging over $\{0,1\}^n$ are (S,ε)-indistinguishable if for every circuit C of size $\leq S$ we have

$$\left|\mathbf{Pr}[C(X)=1] - \mathbf{Pr}[C(Y)=1]\right| \leq \varepsilon.$$

Proof sketch of Lemma 6.5. Given a (S,δ)-hard function f, we first find a set H as in Impagliazzo's Lemma, such that f is $(S',1/2-\alpha)$-hard with respect to the uniform distribution on H, where $S' = S\cdot\mathrm{poly}(\alpha,\delta)$. Then we consider the distributions $(x,f(x))$ and $(x,F_H(x))$, for uniformly distributed x, and we prove that they are $(S'-O(1),\alpha\delta)$-indistinguishable. From this point, it is not hard to show, using a hybrid argument, that the distributions

$$(x_1,\ldots,x_k,f(x_1),\ldots,f(x_k))$$

and

$$(x_1,\ldots,x_k,F_H(x_1),\ldots,F_H(x_k))$$

are $(S'-O(1),k\alpha\delta)$-indistinguishable. Suppose now that g is a function computable in size s and that A is a circuit of size S'' such that

$$\begin{aligned}\mathbf{Pr}[A(x_1,\ldots,x_k) &= g(f(x_1),\ldots,f(x_k))] \\ &> \mathbf{Pr}[A(x_1,\ldots,x_k) = g(F_H(x_1),\ldots,F_H(x_k))] + k\cdot\alpha\delta.\end{aligned}$$

Define the circuit

$$C(x_1,\ldots,x_k,b_1,\ldots,b_k) = A(x_1,\ldots,x_k) \oplus g(b_1,\ldots,b_k)$$

of size $S'' + s + O(1)$ showing that the above two distributions are not $(S'' + s + O(1), k\alpha\delta)$-indistinguishable. It is possible to choose $S'' = S \cdot \text{poly}(\alpha, \delta)$ so that this is a contradiction. □

Lemma 6.5, together with Lemma 6.4, is sufficient to provide amplification of hardness within NP for problems whose characteristic function is balanced.

Lemma 6.6. Let $f : \{0,1\}^n \to \{0,1\}$ be a balanced (S, δ)-hard function. Then for every ε there is a $k = \text{poly}(1/\varepsilon, 1/\delta)$ and a monotone $g : \{0,1\}^k \to \{0,1\}$ computable by a circuit of size $O(k)$ such that if we define

$$h(x_1, \ldots, x_k) = g(f(x_1), \ldots, f(x_k)),$$

we have that h is $(S \cdot \text{poly}(\varepsilon, \delta), 1/2 - \varepsilon)$-hard.

Proof. Apply Lemma 6.4 and find a $k = \text{poly}(1/\varepsilon, 1/\delta)$ and a function $g : \{0,1\}^k \to \{0,1\}$ such that for every set H of size $\delta 2^n$ and every A we have

$$\mathbf{Pr}_{x_1,\ldots,x_k}[A(x_1, \ldots, x_k) = g(F_H(x_1), \ldots, F_H(x_k))] \leq \frac{1}{2} + \frac{\varepsilon}{2}$$

Apply Lemma 6.5 with $\alpha = \varepsilon\delta/2k$ to find a set H such that for every circuit A of size at most $S \cdot \text{poly}(\alpha, \delta) - s = S \cdot \text{poly}(\varepsilon, \delta)$ we have

$$\begin{aligned}\mathbf{Pr}[A(x_1, \ldots, x_k) &= g(f(x_1), \ldots, f(x_k))] \\ &\leq \mathbf{Pr}[A(x_1, \ldots, x_k) = g(F_H(x_1), \ldots, F_H(x_k))] + \frac{\varepsilon}{2}.\end{aligned}$$

Combining the two expressions, we have that for every circuit A of size at most $S \cdot \text{poly}(\varepsilon, \delta)$

$$\mathbf{Pr}[A(x_1, \ldots, x_k) = g(f(x_1), \ldots, f(x_k))] \leq \frac{1}{2} + \varepsilon.$$ □

Some extra work is needed to remove the assumption that the funtion is balanced and to optimize the constants. O'Donnell final result is the following.

Theorem 6.7. (O'Donnell) Suppose that for every language L in NP we have $(L,\mathcal{U}) \in \mathrm{Heur}_{1/2-1/n^{.33}}\mathrm{P/poly}$. Then for every polynomial p and for every language L in NP we have

$$(L,\mathcal{U}) \in \mathrm{Heur}_{1/p(n)}\mathrm{P/poly}.$$

The result was improved by Healy *et al.* [40], but only for balanced languages (i.e., for languages whose characteristic function is balanced on every input length).

Theorem 6.8. (Healy *et al.*) Suppose that for every balanced language L in NP there is a polynomial p such that $(L,\mathcal{U}) \in \mathrm{Heur}_{1/2-1/p(n)}\mathrm{P/poly}$. Then for every polynomial p and for every balanced language L in NP we have

$$(L,\mathcal{U}) \in \mathrm{Heur}_{1/p(n)}\mathrm{P/poly}.$$

Trevisan [70,72] proves weaker results for the uniform HeurBPTIME classes. Specifically, Trevisan proves that there is a constant c such that if $(\mathrm{NP},\mathcal{U}) \subseteq \mathrm{Heur}_{1/2-1/(\log n)^c}\mathrm{BPP}$ then, for every polynomial p, $(\mathrm{NP},\mathcal{U}) \in \mathrm{Heur}_{1/p(n)}\mathrm{BPP}$.

Indeed, the actual result is slightly stronger.

Theorem 6.9. (Trevisan) Suppose that for every language L in NP there is a polynomial time randomized algorithm A such that for every n

$$\mathbf{Pr}_{x\sim U_n;\text{coin tosses of } A}[A(x) \neq L(x)] \leq \frac{1}{2} + \frac{1}{(\log n)^c}.$$

Then, for every polynomial p, $(\mathrm{NP},\mathcal{U}) \in \mathrm{Heur}_{1/p(n)}\mathrm{BPP}$.

Note that the assumption in the theorem is (possibly) weaker than $(\mathrm{NP},\mathcal{U}) \subseteq \mathrm{Heur}_{1/2-1/(\log n)^c}\mathrm{BPP}$, which requires

$$\mathbf{Pr}_{x\sim U_n}\left[\mathbf{Pr}_{\text{coin tosses of } A}[A(x) \neq L(x)] > \frac{1}{4}\right] \leq \frac{1}{2} + \frac{1}{(\log n)^c}.$$

7

Worst-Case Versus Average-Case and Cryptography

The results on hardness amplification from Chapter 6 indicate that the notion of average-case hardness is very robust with respect to the hardness parameter. Namely, it is just as hard to solve hard problems in (NP,$\mathcal{U}$) on slightly more than half their inputs as it is to solve them on a $1 - 1/\text{poly}(n)$ fraction of inputs. It is reasonable to ask whether this connection can be pushed to the extreme: Is it the case that solving problems in (NP,$\mathcal{U}$) on slightly more than half their inputs is no easier than solving them on *all* inputs? In other words, are there problems in (NP,$\mathcal{U}$) whose tractability would imply that NP $\subseteq$ BPP?

A related and fundamental question in cryptography is whether the security of various cryptographic primitives can be reduced to a reasonable worst-case complexity theoretic assumption, such as NP $\not\subseteq$ BPP. This question has not been settled yet, and there is contrasting evidence about the possibility of such a connection. In this section we review and explain several results related to this topic. As we shall see, at the heart of the question of basing cryptography on a worst-case assumption is the connection between worst-case and average-case complexity.

Various cryptographic tasks require cryptographic primitives of seemingly different strength. Here, we focus on the worst-case

assumptions necessary for the existence of one-way functions (equivalently, symmetric key cryptography) and public key encryption.

Since under the assumption NP $\subseteq$ BPP no one-way functions exist, a worst-case assumption necessary for the existence of one-way functions must be at least as strong as NP $\not\subseteq$ BPP. Is this assumption sufficient for the existence of one-way functions? If it is not, is it possible to base the existence of one-way functions on a possibly relaxed, but still reasonable, worst-case complexity assumption?

Assuming the worst-case intractability of certain promise problems on lattices, it is possible to obtain provably secure constructions of cryptographic one-way functions, as well as seemingly stronger primitives such as collision-resistant hash functions and public key encryption schemes. However, all known worst-case intractable problems that yield secure cryptographic primitives are both in NP and coNP, and are thus unlikely to be NP hard.[1]

At this point, it is an open question whether the average-case tractability of (NP,$\mathcal{U}$) would imply that NP $\subseteq$ BPP, and whether any form of cryptography can be based on the assumption NP $\not\subseteq$ BPP. In this section we review evidence that points to some difficulties in establishing such connections.

7.1 Worst-case to average-case reductions

What do we mean when we say that the existence of one-way functions can be based on the assumption NP $\not\subseteq$ BPP? The most general interpretation would be to say that there exists a proof of the statement "NP $\not\subseteq$ BPP implies that one-way functions exist." At this point no such proof is known; however, it is difficult to rule out the existence of a proof, for that would imply that either "NP $\not\subseteq$ BPP" or "one-way functions exist" would not be provable. One plausible interpretation of the claim that the existence of one-way functions requires assumptions stronger than NP $\subseteq$ BPP would be to say that any "plausible" way to obtain a worst-case algorithm for SAT (or some other NP-complete

[1] The worst-case assumption that statistical zero knowledge contains intractable problems, which seems to be much stronger than NP $\not\subseteq$ BPP, is known to imply the existence of infinitely often one-way functions, a primitive object seemingly weaker than the one-way function [62]. This primitive does not appear to have any useful applications.

problem) from an imagined inverter for the universal one-way function fails, or at least violates some reasonable assumption.

To see what we mean by "plausible," let us see how a possible proof of the claim might go. Generally such proofs are carried out by reduction; namely, there is an efficiently computable procedure that maps candidate inverters for the one-way function to algorithms for SAT. Moreover, the reductions typically use the one-way function inverter as a black box only. Such a reduction can be modeled as an efficient oracle procedure R that, when given oracle access to an average-case inverter for the one-way function, solves SAT correctly on almost all instances. With this in mind, the notion that one-way functions can be based on the assumption "NP $\not\subseteq$ BPP" can be liberally interpreted as the existence of a reduction R of the form described above.

We would also like to consider the possibility that one-way functions can be based on stronger assumptions. This motivates the notion of a worst-case to average-case reduction. First, we define the notion of an "inversion oracle" for a one-way function.

Definition 7.1. (inversion oracle) Let $\{f_n : \{0,1\}^n \to \{0,1\}^*\}$ be a family of functions. An inversion oracle for $\{f_n\}$ with error $\delta(n)$ is a family of (possibly randomized) functions $\{I_n : \{0,1\}^* \to \{0,1\}^n\}$ such that for all n,

$$\mathbf{Pr}_{x \sim U_n, I_n}[I_n(f_n(x)) \notin f_n^{-1}(f_n(x))] \leq \delta(n).$$

Thus, if there is an efficiently computable inversion oracle for f with inverse polynomial error, then f is not strongly one way.

Definition 7.2. (worst-case to average-case reduction) A *worst-case to average-case reduction* from a language L to inverting a family of functions $\{f_n\}$ with average-case error $\delta(n)$ is an oracle procedure R such that for all inversion oracles I with error $\delta(n)$, all sufficiently large n, and all x of length n,

$$\mathbf{Pr}_{R,I}[R^I(x) \neq L(x)] < \frac{1}{3}.$$

The reduction is called *non-adaptive* if the reduction makes all its queries in parallel, that is, each query is independent of answers to previous queries.

If the function f were not one way, the inversion oracle could be implemented by an efficient algorithm, and the reduction would give an efficient algorithm for L. Thus, a worst-case to average-case reduction can be viewed as a fairly general tool for establishing a connection between the average-case complexity of inverting f and the worst-case complexity of L.

In a similar fashion, we can define worst-case to average-case reductions for other primitives in average-case complexity, in particular, distributional decision problems and distributional search problems (of which one-way functions are a special case). The only part of the definition that differs for these primitives is the notion of an inversion oracle, which we call "approximate oracle" in this context. For illustration we state the definition for deterministic oracles, and for decision problems only.

Definition 7.3. Let L be a language and $\mathcal{D}$ an ensemble of distributions. An *approximate oracle* for $(L,\mathcal{D})$ with error $\delta(n)$ is a function $A : \{0,1\}^* \to \{0,1,\perp\}$ such that for all n,

$$\mathbf{Pr}_{x\sim D_n}[A(x) \neq L(x)] < \delta(n).$$

The approximate oracle is *errorless* if for all x, $A(x) \in \{L(x), \perp\}$.

A *worst-case to average-case reduction* with error $\delta(n)$ from L to $(L',\mathcal{D})$ is an efficient oracle procedure R such that for all approximate oracles A with error $\delta(n)$, all sufficiently large n, and all x of length n, $\mathbf{Pr}_R[R^A(x) \neq L(x)] < \frac{1}{3}$.

Thus, if $(\mathrm{BH},\mathcal{U}^{\mathrm{BH}})$ has an efficiently computable approximate oracle, then $(\mathrm{NP},\mathrm{PSAMP}) \subseteq \mathrm{HeurBPP}$; if the oracle is errorless, then $(\mathrm{NP},\mathrm{PSAMP}) \subseteq \mathrm{AvgZPP}$. Assuming $\mathrm{NP} \not\subseteq \mathrm{BPP}$, the existence of a worst-case to average-case reduction from SAT to $(\mathrm{BH},\mathcal{U}^{\mathrm{BH}})$ implies that $(\mathrm{NP},\mathrm{PSAMP}) \not\subseteq \mathrm{HeurBPP}$ (or $(\mathrm{NP},\mathrm{PSAMP}) \not\subseteq \mathrm{AvgZPP}$, if the reduction only works with respect to errorless oracles).

This definition of "worst-case to average-case reduction" models the framework used to establish the amplification of hardness results from Chapter 6. Also, in the extreme case $\delta = 0$, the definition becomes the standard notion of reducibility between worst-case problems.

Alternative definitions: The notion of "worst-case to average-case reduction" attempts to capture a reasonable class of possible approaches for basing average-case complexity and cryptography on NP-hardness. We wish to stress, however, that the definition is by no means canonical and that it is natural to consider certain variants. For simplicity we focus on Definition 7.3.

One alternative to Definition 7.3 is to consider generic procedures that, given oracle access to *any* worst-case hard language L, produce an average-case hard language $(L', \mathcal{D})$. For such a procedure A to be useful for NP languages, it should be the case that A itself is an NP procedure with access to an oracle. This notion is interesting because such procedures exist in higher complexity classes such as PSPACE and EXP, where they are used to establish worst-case to average-case connections. The amplification results of Chapter 6 are also of this type. Viola [75] (see also [74]) shows that no such oracle procedure exists in NP, and even in the polynomial hierarchy (unless $(\mathrm{NP}, \mathrm{PSAMP}) \not\subseteq \mathrm{HeurP/poly}$, in which case A exists trivially).

In summary, Viola's result shows that any worst-case to average-case reduction in NP must use specific properties of the worst-case language it is reducing from. Indeed, the worst-case to average-case reductions of Ajtai, Micciancio, and Regev heavily exploit properties that are specific to lattices.

A serious limitation of Definition 7.3 is that it does not impose any computational restriction on the average-case oracle.[2] In reality, to base average-case complexity on NP-hardness, the reduction need only consider candidate average-case oracles that can be implemented in BPP. This intriguing type of a reduction is called a "BPP-class black-box reduction" by Gutfreund and Ta-Shma [37]: As in Definition 7.3, the reduction only obtains oracle (black box) access to the average-case

[2] In fact, all results presented in this section hold for Σ_2 oracles, and in some cases for NP oracles.

solver, but is allowed to behave arbitrarily if the oracle cannot be implemented in BPP. Gutfreund, Shaltiel, and Ta-Shma [36, 37] show an interesting setting in which BPP-class black box reductions are provably more powerful than ordinary worst-case to average-case reductions (under reasonable assumptions). However, it is not known whether such reductions can be used to base average-case complexity for NP and cryptography on NP-hardness.

It is of course possible to further relax the definition and allow the reduction non-black box access to an implementation of the inversion oracle. Little is known about the power of such a setting.

7.2 Permutations and range-computable functions

What is the hardest language L for which we can expect to have a worst-case to average-case reduction from L to inverting some one-way function? Let us look at some simple cases first.

First, let us consider the case of a reduction R from L to a one-way permutation $f : \{0,1\}^n \to \{0,1\}^n$. Then it is not difficult to see that L must be in AM $\cap$ coAM (NP $\cap$ coNP if the reduction is deterministic). The situation is completely analogous for L and $\overline{L}$, so it is sufficient to prove that $L \in$ AM. A simple two-round protocol for deciding membership in L works as follows: In the first round, the verifier sends the coins used by the reduction to the prover. In the second round, the prover sends the verifier a transcript that describes the computation on R when given access to an oracle that inverts f on all inputs. When R makes oracle query q, the honest prover answers with the unique a such that $f(a) = q$. The verifier can check that all the answers provided by the prover are consistent with its queries, thus forcing the prover to perfectly simulate a computation of R when given oracle access to an inverter for f. At the end of the interaction, the verifier accepts iff the transcript provided by the prover is an accepting transcript for R.

It follows that the average-case hardness of any one-way permutation can be based, at best, on the worst-case hardness of some problem in AM $\cap$ coAM. Thus, there appears to be no hope of basing the hardness of any cryptosystem that requires one-way permutations on the assumption NP $\not\subseteq$ BPP.

7.2.1 *k*-to-one functions

A permutation is a function that is both onto and one-to-one; Akavia *et al.* [9] consider what happens when the function $f : \{0,1\}^{n+\log k} \to \{0,1\}^n$ is k-to-one, namely every element in $\{0,1\}^n$ has exactly k pre-images under f. The crucial difference between the cases $k = 1$ and $k > 1$ is that when $k = 1$, the function f admits a unique inverting oracle, while for $k > 1$ there are many such oracles. To illustrate the significance of this, let us see what happens when the above protocol for permutations is applied to a two-to-one function f. Since the number of inverting oracles for f is now doubly exponential in n, it may be the case that for every choice of randomness by the reduction, there exists some inversion oracle that makes the reduction output the incorrect answer. A cheating prover can then force the verifier to output the incorrect answer by using this inversion oracle in its simulation.

The solution of Akavia *et al.* is to force the prover to commit to a particular oracle that is independent of the randomness used by the reduction. Let us first illustrate this with the case $k = 2$. Then it is easy to modify the protocol for L so that the prover is always forced to simulate interaction with the "smallest" inverting oracle for f: This is the inverter that, on input q, always answers with the lexicograpically smaller pre-image of q under f. To check correctness, for every query q the verifier always asks to see *both* pre-images of q, and always uses the smaller of the two values in its simulation of the reduction. It is straightforward that this argument works for any k up to $\text{poly}(n)$.

For values of k larger than $\text{poly}(n)$, it is infeasible to ask the prover to provide a complete list of pre-images for each query. Instead, the prover is forced to provide a *random* pre-image, which is independent of the randomness used by the reduction. Thus, the prover will simulate the interaction of R with a random inverter. Let us outline how such a random pre-image might be obtained. The random inverter that the proof system intends to simulate is the following one: For each possible query q, choose a random hash function h mapping n bits to slightly fewer than $\log_2(k/s)$ bits, where $s = \text{poly}(n)$. With high probability, the size of the set $S = h^{-1}(0) \cap f^{-1}(q)$ is about s. Out of all the elements

of S, choose the lexicographically smallest one (and if S is empty, choose an arbitrary inverse of q).

As a first attempt, consider this proof system for simulating the inverter on a query q: The verifier chooses a random hash function h, asks the prover for a complete list of members of S, and chooses the lexicographically smallest one. Notice that no prover can include fictitious members of S in its list because membership in S is an efficiently verifiable property. Therefore, provers can only cheat in a "one-sided" manner: A cheating prover can attempt to omit members of S, but never claim fictitious members of S.

A cheating prover may, of course, fool the verifier by claiming that, say, S is empty. The verifier knows that the size of S must be approximately s, so the verifier can protect against such an attack by rejecting all sets S whose size deviates substantially from s. The problem is that the cheating prover may fool the verifier even by omitting a *single* entry of S, namely the lexicographically smallest one. Hence the verifier must ensure that the prover has not omitted even a single element of S.

This appears impossible to achieve in general, as deviation bounds on the size of S only guarantee that S will have roughly the expected number of elements. Instead, Akavia *et al.* consider what happens when we fix the randomness used by the reduction and execute this protocol $t = \mathrm{poly}(n)$ times independently in parallel. Let S_i denote the set S resulting from the ith run of the protocol.

Now suppose that for every potential query q, it can be guaranteed that in a $1 - \varepsilon$ fraction of the t protocol runs, the prover provides the correct set S_i. Then at least a $1 - \varepsilon$ fraction of the protocol runs provide a correct answer to the first query asked by the reduction; out of those, a $1 - \varepsilon$ fraction of runs provide a correct answer to the second query, and so on. If the verifier asks ℓ queries, then a $(1 - \varepsilon)^\ell$ fraction of runs will have all their queries answered correctly. By choosing ε small enough, it can be ensured that a random run simulates the reduction correctly with high probability.

Therefore, the main task is to design a verifier test that ensures that a $1 - \varepsilon$ fraction of the t protocol runs yield the correct set S_i. The crucial point is that in order to make the verifier fail with probability ε, a cheating prover must now omit at least εt elements from the union

of sets $S_1 \cup \cdots \cup S_t$.[3] For $t \gg s/\varepsilon^2$, εt becomes a significant deviation from st, the expected size of this union. Statistically, we know that with high probability,

$$\big||S_1 \cup \cdots \cup S_t| - st\big| < \varepsilon t/2$$

so if the verifier checks that

$$\sum_{i=1}^{t} |\text{prover's claim for } S_i| \geq st - \varepsilon t/2$$

the honest prover will pass this check with high probability. On the other hand, this severely limits the power of a cheating prover: If any prover omits more than εt elements from $S_1 \cup \cdots \cup S_t$, then

$$\begin{aligned}\sum_{i=1}^{t} |\text{prover's claim for } S_i| &< |S_1 \cup \cdots \cup S_t| - \varepsilon t \\ &< (st + \varepsilon t/2) - \varepsilon t \\ &< st - \varepsilon t/2,\end{aligned}$$

and the verifier rejects. Notice that the soundness of this protocol relies on the fact that the power of a cheating prover is one sided: A cheating prover can only understate, but never overstate the size of the sets S_i.

One additional condition that must be ensured is that the sets S_i are non-empty for most i, for otherwise not even the honest prover can correctly simulate the inverter for f. This can be achieved by an appropriate choice of parameters.

Size-computable, size-approximable, and size-certifiable functions: A family of functions $f_n : \{0,1\}^n \to \{0,1\}^*$ is *size computable* if there is an efficient algorithm that on inputs n and y runs in time polynomial in n and outputs the number $|f_n^{-1}(y)|$. The k-to-one functions considered above can be viewed as a special case of size-computable functions. If the algorithm outputs an approximation of $|f_n^{-1}(y)|$ within an arbitrary factor that is inverse polynomial in n, the family is called *size approximable*. If the algorithm is non-deterministic, the family is called *size certifiable*. The protocol

[3] By convention we assume that the sets are pairwise disjoint.

of Akavia *et al.* naturally extends to the case of size-computable, size-approximable, and size-certifiable functions.

Theorem 7.4. (Akavia *et al.*) Suppose there exists a worst-case to average-case reduction from language L to inverting a size-approximable or size-certifiable family of functions $\{f_n\}$. Then $L \in \mathrm{AM} \cap \mathrm{coAM}$.

An example of a size-certifiable family is the family of functions

$$f_n(p,q) = \begin{cases} p \cdot q & \text{if } p \text{ and } q \text{ are } \lfloor n/2 \rfloor\text{-bit primes,} \\ 0 & \text{otherwise.} \end{cases}$$

It is widely believed that this family of functions is weakly one way. However, Theorem 7.4 shows that the problem of inverting this family is unlikely to be NP-hard.

7.3 General one-way functions and average-case hard languages

Theorem 7.4 can be interpreted as evidence that it may not be possible to base the hardness of one-way functions on an NP-complete problem. The requirement that the family $\{f_n\}$ be range certifiable may appear to be a technical one, and it is often the case that the existence of one-way functions satisfying some additional technical requirement is equivalent to the existence of general one-way functions.

We will argue that this interpretation of Theorem 7.4 is mistaken. Observe that the protocol of Akavia *et al.* in fact simulates a run of the reduction interacting with a *worst-case* inversion oracle for f_n, not an average-case one; thus it shows that even the more difficult problem of inverting $y = f_n(x)$ on *every* output y is unlikely to be NP-hard.

On the other hand, we do know of one-way functions that are NP-hard to invert in the worst case. For instance, consider the function f that maps a CNF φ and an assignment a for φ to $(\varphi, \varphi(a))$. A worst-case inversion algorithm for f solves the search version of SAT. Naturally, we do not interpret this as saying that "f is a one-way function that is NP-hard to invert," because it may well be the case that even

though f is NP-hard to invert on all inputs, it is invertible on most inputs. (This is in fact true for many natural choices of distribution on inputs.)

Thus, if it is indeed the case that the hardness of inverting one-way functions cannot be based on an NP-complete problem, the argument must use the fact that the assumed reduction from the NP-complete problem to the inversion oracle works correctly with respect to an *average-case* inversion oracle, not only for a worst-case one.

At this point it is not known whether such reductions exist in general. The techniques described in the previous section can be viewed as partial progress toward a negative result that are obtained by putting restrictions on the type of one-way function under consideration. In this section we present a different approach that allows for general one-way functions but places restrictions on the type of reduction used to establish the worst-case to average-case equivalence. In contrast to Theorem 7.4, some of the results presented below make essential use of the fact that the one-way function must be hard to invert on average.

We begin by looking at the connection between worst-case and average-case hardness for languages, rather than functions. In particular, we focus on the relation between the conjectures NP $\not\subseteq$ BPP and (NP,$\mathcal{U}$) $\not\subseteq$ HeurBPP.

7.3.1 The Feigenbaum-Fortnow approach

What can a worst-case to average-case reduction from a language L to a distributional NP problem $(L',\mathcal{U})$ look like?

To begin with, we observe that if the reduction is deterministic, then L must be in P: For any $x \in \{0,1\}^*$, the answer produced by the reduction on input x must be independent of the choice of average-case oracle for L'. One such average-case oracle is the oracle that agrees with L' on all the strings that are not queried by the reduction on input x, and answers $\perp$ on all the other queries. From the point of view of the reduction, however, this oracle is indistinguishable from the oracle that answers $\perp$ on every query. Therefore, an efficient algorithm for L can be obtained by simulating the reduction on input x with access to an oracle that always answers $\perp$.

It follows that any non-trivial worst-case to average-case reduction must make randomized queries to the average-case oracle. Feigenbaum and Fortnow [27] consider the case in which the reduction is non-adaptive and the distribution of every query made by the reduction on input x of length n is uniform in $\{0,1\}^{n'}$ for some $n' = \text{poly}(n)$. Reductions of these type are called *locally random reductions*. The reason such reductions are interesting is that they provide a natural way of establishing a worst-case to average-case connection: If the reduction asks q queries, then any average-case oracle that is $1/4qn'$-close to L' with respect to the uniform distribution is indistinguishable from L' itself from the point of view of the reduction with probability $\frac{3}{4}$. Thus, if there exists a locally random reduction from L to L', and L is hard in the worst-case, then L' is hard to solve on more than a $1 - 1/4qn'$-fraction of inputs. Locally random reductions have been used to establish worst-case to average-case connections in settings other than NP.

Feigenbaum and Fortnow essentially rule out locally random reductions as a tool for establishing worst-case to average-case connection for all of NP. More precisely, they show that if there exists a locally random reduction from a language L to a language L' in NP, then it must be that L is in NP/poly $\cap$ coNP/poly. In particular, L is unlikely to be NP-hard: If L is NP-hard, then NP is contained in coNP/poly, and the polynomial hierarchy collapses to the third level.

To prove this, Feigenbaum and Fortnow give a way to simulate the reduction (on input x) by an AM proof system that uses polynomial length non-uniform advice. The outcome of the simulation then determines whether x is a "yes" or a "no" instance of L. Thus, the protocol can be used to determine membership in both L and $\overline{L}$. An AM proof system with advice can be turned into a non-deterministic circuit, giving the conclusion $L \in$ NP/poly $\cap$ coNP/poly.

The Feigenbaum-Fortnow protocol: Let R be a locally random reduction from L to $L' \in$ NP. Suppose that on an input of length n, R makes k queries, each of which is uniformly distributed in $\{0,1\}^{n'}$. Without loss of generality, assume that R is correct with very high probability (say $1 - 1/k^3$) over its random coins.

We show an interactive protocol for membership in L. The protocol for $\overline{L}$ is identical except that it inverts the answers given by R.

The non-uniform advice used by the protocol will be the value $p = \mathbf{Pr}_{y \sim \{0,1\}^{n'}}[y \in L']$.

The protocol: On input $x \in \{0,1\}^n$,

(1) **Verifier:** Run $R(x)$ independently $m = 64k^2 \log k$ times to generate m sets of queries $(y_1^1, \ldots, y_k^1), \ldots, (y_1^m, \ldots, y_k^m)$. Send all queries to the prover.

(2) **Prover:** For each y_i^j, respond by saying whether $y_i^j \in L'$. Accompany each claim that $y_i^j \in L'$ by an NP-certificate for y_i^j.

(3) **Verifier:** Accept if all of the following conditions hold:

   (a) $R(x)$ accepts in all m iterations using the answers provided by the prover,

   (b) All certificates sent by the prover are valid, and

   (c) For every $1 \leq j \leq k$, at least $pm - m/2k$ of the queries $y_j^1, \ldots, y_j^m$ are answered "yes."

If $x \in L$ and the prover follows the protocol, then $R(x)$ accepts in all m iterations with high probability, and the verifier accepts provided condition 3(c) is satisfied. Note that for each fixed j, the strings $y_j^1, \ldots, y_j^m$ are independent and uniformly distributed in $\{0,1\}^{n'}$, and each one has probability p of being a yes instance. By Chernoff bounds, with probability at least $1/4k$, at least $pm - 4\sqrt{m \log k} > pm - m/2k$ of them are yes instances. By a union bound with probability $\frac{3}{4}$, this is satisfied for all j and condition 3(c) holds.

If $x \notin L$, to make the verifier accept, the prover must send an erroneous answer in every one of the m runs of $R(x)$, so in particular there must be at least m errors among the prover's answers. All the erroneous answers of the prover must be yes instances on which it answers no (if the prover tries to cheat the other way, it would not be able to provide certificates). In particular, there must be some j such that among the queries $y_j^1, \ldots, y_j^m$ at least m/k are answered no even though they were yes instances. By a Chernoff bound as above, it is unlikely that there

are more than $pm + 4\sqrt{m \log k}$ yes instances among $y_j^1, \ldots, y_j^m$, so the prover is giving at most $pm + 4\sqrt{m \log k} - m/k < pm - m/2k$ "yes" answers for $y_j^1, \ldots, y_j^m$. Then the verifier rejects with high probability in step 3(c).

7.3.2 Arbitrary non-adaptive reductions

For the result of Feigenbaum and Fortnow, it is not necessary that the distribution of each query made by the reduction be uniform over $\{0,1\}^{n'}$, but it is essential that the marginal distribution of queries made by the reduction be independent of the reduction's input. This restriction is quite strong, and in this sense, the result is extremely sensitive: If one modifies the distribution of queries even by an exponentially small amount that depends on the input, all statistical properties of the reduction are preserved, but one can no longer draw the conclusion that $L \in \mathrm{NP/poly} \cap \mathrm{coNP/poly}$.

Bogdanov and Trevisan [15] show that the conclusion of Feigenbaum and Fortnow holds in a more general setting. They show that the existence of any non-adaptive worst-case to average-case reduction from L to an arbitary problem $(L', \mathcal{D})$ in $(\mathrm{NP}, \mathrm{PSAMP})$ implies that L is in $\mathrm{NP/poly} \cap \mathrm{coNP/poly}$, with no restriction on the distribution of queries made by the reduction. In particular, the queries made by the reduction are allowed to depend arbitrarily on the input x. This formulation extends the result of Feigenbaum and Fortnow in two directions: First, et allows for a more general class of worst-case to average-case reductions; second, et allows average-case complexity to be measured with respect to an arbitrary samplable distribution, not only the uniform distribution.

Theorem 7.5. (Bogdanov and Trevisan) Suppose that there exists a non-adaptive worst-case to average-case reduction from a language L to a decision problem $(L', \mathcal{D})$ in $(\mathrm{NP}, \mathrm{PSAMP})$. Then $L \in \mathrm{NP/poly} \cap \mathrm{coNP/poly}$.

The proof of Bogdanov and Trevisan uses essentially the fact that the reduction is correct when given access to an arbitrary average-case oracle for $(L', \mathcal{D})$. The idea of the proof is again to simulate the

reduction querying an average-case oracle for $(L', \mathcal{D})$ with an AM protocol using advice. Observe that the Feigenbaum-Fortnow protocol works for arbitrary non-adaptive reductions whenever it is given as auxiliary input the probability p_x that a random query made by the reduction on input x is a "yes" instance of L' according to distribution $\mathcal{D}$. For a general reduction, however, the value p_x cannot be provided as advice for the protocol because it may depend on the particular input x.

The idea of Bogdanov and Trevisan is to use a different protocol to compute the value p_x, then use the Feigenbaum-Fortnow protocol for membership in L using the value p_x as auxiliary input. Initially, a weaker version of the theorem is proved where $\mathcal{D}$ is the uniform distribution. To begin with, let us allow the distribution of queries made by the reduction to depend on x, but restrict it to be "α-smooth": We assume that every query y is generated with probability at most $\alpha \cdot 2^{-|y|}$, where α is a constant. Suppose that, given a *random* query y, we could force the prover to reveal whether or not $y \in L'$. Then by sampling enough such queries y, we can estimate p_x as the fraction of "yes" queries made by the reduction. But how do we force the prover to reveal if $y \in L'$? The idea is to hide the query y among a sequence of queries $z_1, \ldots, z_k$ for which we *do* know whether $z_i \in L'$, in such a way that the prover cannot tell where in the sequence we hid our query y. In such a case, the prover is forced to give a correct answer for y, for if he were to cheat he would not know where in the sequence to cheat, thus would likely be caught.

The problem is that we do not know a specific set of queries z_i with the desired property. However, the strings z_i were chosen by sampling independently from $\mathcal{D}$, then with high probability $pk \pm O(\sqrt{k})$ of these queries will end up in L', where p is the probability that a string sampled from $\mathcal{D}$ is in L'. Since p depends only on the length of x but not on x itself, it can be given to the verifier non-uniformly. This suggests the following verifier strategy: Set $k = \omega(\alpha^2)$, generate k uniformly random queries $z_1, \ldots, z_k$ of length n, hide y among $z_1, \ldots, z_k$ by inserting it at a random position in the sequence, send all the queries to the prover and ask for membership in L' together with witnesses that at least $pk - O(\sqrt{k})$ queries belong to L'. Then with high probability, either the verifier rejects or the answer about membership of y in L' is likely

correct. Intuitively, a cheating prover can give at most $O(\sqrt{k})$ wrong answers. The prover wants to use this power wisely and assign one of these wrong answers to the query y. However, smoothness ensures that no matter how the prover chooses the set of $O(\sqrt{k})$ queries to cheat on, it is very unlikely that the query y falls into that set.

For a reduction that is not smooth, it is in general impossible to hide a query y among random queries from $\mathcal{D}$ using the above approach. However, suppose that the verifier had the ability to identify queries y that occur with probability $\geq \alpha \cdot 2^{-|y|}$; let us call such queries "heavy" and the other ones "light." The fraction of heavy queries in $\mathcal{D}$ is at most $1/\alpha$. Suppose also that the prover answers all light queries correctly. The prover can then certify membership in L as follows: If the query made by the reduction is heavy, pretend that the average-case oracle answered $\perp$, otherwise use the answer provided by the prover. This process simulates exactly a run of the reduction when given access to an average-case oracle that agrees with L' on all the light queries, and answers $\perp$ on all the heavy queries. In particular, the oracle agrees with L' on a $1 - 1/\alpha$ fraction of strings, so the reduction is guaranteed to return the correct answer.

In general, the verifier cannot identify which queries made by the reduction are heavy and which are light. The last element of the construction by Bogdanov and Trevisan is an AM protocol with advice that accomplishes this task.

The case of a general samplable distribution $\mathcal{D}$ can be reduced to the case when $\mathcal{D}$ is the uniform distribution using Theorem 5.2, observing that the reduction in the proof is indeed non-adaptive.

7.3.3 Search problems and one-way functions

Theorem 7.5 shows that non-adaptive worst-case to average-case reductions from an NP-hard problem to decision problems in $(\mathrm{NP}, \mathrm{PSAMP})$ are unlikely to exist. How about reductions to search problems? Using the fact that search-to-decision reduction described in Section 4.2 is non-adaptive, we can conclude that non-adaptive reductions from NP-hard problems to distributional search problems in NP are also unlikely to exist.

A case of special interest is when the distributional search problem is inverting a one-way function: If there exists a non-adaptive worst-case to average-case reduction from a language L to a family of functions $\{f_n\}$, then $L \in \mathrm{NP/poly} \cap \mathrm{coNP/poly}$. Using a more refined argument for the case of one-way functions, Akavia *et al.* obtain a simulation of the reduction by an AM protocol without advice:

Theorem 7.6. (Akavia *et al.*) Suppose that there exists a non-adaptive worst-case to average-case reduction from language L to inverting a family of functions $\{f_n\}$. Then $L \in \mathrm{AM} \cap \mathrm{coAM}$.

7.4 Public key encryption

Do there exist public key encryption schemes whose security can be based on the assumption $\mathrm{NP} \not\subseteq \mathrm{BPP}$? Since public key encryption schemes are harder to design than one-way functions, we expect that this question should be only harder to answer in the affirmative than the question whether one-way functions follow from the assumption $\mathrm{NP} \not\subseteq \mathrm{BPP}$. Conversely, the lack of cryptographic primitives based on NP hardness assumptions should be easier to explain in the public key setting than in the symmetric key setting.

As in the case of one-way functions, we interpret the question whether public key encryption can be based on the assumption that $\mathrm{NP} \not\subseteq \mathrm{BPP}$ as asking for the existence of an efficiently computable reduction that converts any adversary that breaks the encryption scheme into an algorithm for SAT. By an encryption scheme, we mean a collection consisting of a key generation algorithm G, an encryption algorithm E, and a decryption algorithm D (all randomized) such that

(1) algorithm G takes as input a hardness parameter n, runs in time polynomial in n, and produces a pair of keys: the public key pk and the secret key sk;
(2) algorithm E takes as inputs a hardness parameter n, a public key pk, and a bit b to be encrypted, runs in time polynomial in n, and satisfies the property that for most public keys

pk (obtained by running $G(n)$), the distributions $E(n,pk,0)$ and $E(n,pk,1)$ are computationally indistinguishable (with respect to the parameter n, by an algorithm that takes as auxiliary input n and pk);

(3) algorithm D takes as inputs a hardness parameter n, a secret key sk, and a ciphertext c, runs in time polynomial in n, and satisfies the property that for all b, and most pairs (pk, sk) obtained from $G(n)$, $D(n, sk, E(n, pk, b)) = b$ with probability negligible in n.

The existence of one-bit encryption is sufficient to construct public key encryption schemes for messages of arbitrary length that satisfy very strong notions of security.

As in the case of one-way functions, it is not known in general whether there exists a reduction from SAT to an adversary for some one-bit encryption scheme. However, such reductions can be ruled out under certain restrictions either on the cryptosystem in question or on the way the reduction works.

Goldreich and Goldwasser [33], building upon previous work by Brassard [17] restrict attention to encryption schemes where for all n and pk, the sets $E(n,pk,0)$ ane $E(n,pk,1)$ are disjoint, and moreover the set

$$S = \{(1^n, pk, c) : c \notin E(n,pk,0) \cup E(n,pk,1)\}$$

is in NP (namely, the property that c is a possible ciphertext is efficiently refutable). Goldreich and Goldwasser observe that some, but not all, known one-bit encryption schemes satisfy these properties. They observe that if there is a reduction from a language L to an adversary for an encryption scheme of this type, then $L \in \mathrm{AM} \cap \mathrm{coAM}$. The reason is that the reduction can be simulated by a two-round proof system in which the prover plays the role of a distinguishing oracle for the sets $E(n,pk,0)$ and $E(n,pk,1)$. In the first round, the verifier chooses the randomness to be used by the reduction and sends it to the prover. In the second round, the prover sends a transcript of the reduction interacting with an adversary for the encryption scheme. When the reduction queries the adversary on

input (n, pk, c), there are three possibilities: Either $c \in (n, pk, 0)$, or $c \in (n, pk, 1)$, or $(n, pk, c) \in S$. By assumption, all three of these cases are efficiently certifiable. Therefore, a transcript of the reduction augmented by certificates for the answers made by every query asked by the reduction constitutes a valid and efficiently checkable simulation of the reduction interacting with a distinguishing oracle for one-bit encryption.

The requirement that the sets of possible encryptions of 0 and 1 are disjoint can be somewhat relaxed, and the requirement that the set S is in NP can be substituted by a requirement that the reduction is "smart" – it never queries invalid ciphertexts. Thus, the observation of Goldreich and Goldwasser can be viewed as saying that the NP hardness of one-bit encryption cannot be established via "non-smart" reductions.

Should these arguments be viewed as an indication that public key cryptography cannot be based on NP-hard problems? Observe that the proof systems of Brassard, Goldreich, and Goldwasser do not use the fact that the reduction outputs the correct answer even if it interacts with an average-case distinguisher between the encryptions of 0 and 1. Thus, these are essentially results about the worst-case complexity of breaking encryption, showing that under certain restrictions on the encryption scheme or on the reduction, the hardness of breaking the encryption *in the worst case* is a problem in $\mathrm{NP} \cap \mathrm{coNP}$. However, these restrictions on the encryption scheme or on the reduction cannot be so easily removed. As was shown by Lempel [51], there do exist "encryption schemes" that are NP-hard to break in the worst case, but are tractable to break on average: Namely, the problem "On input $(n, pk, E(n, pk, b))$, find b" is NP-hard in the worst case, but is tractable on average. (Lempel's result generalizes the observation that there exist one-way functions that are NP hard to invert in the worst case but easy to invert on average to the setting of public key cryptography.) Currently, there is no known argument that explains why public key cryptography appears to require worst-case assumptions stronger than $\mathrm{NP} \not\subseteq \mathrm{BPP}$ beyond what is known for one-way functions, that is, symmetric key cryptography.

7.5 Perspective: Is distributional NP as hard as NP?

So far we have focused on negative results regarding connections between the worst-case and average-case complexity of NP. Since these results do not rule out the possiblity that distributional NP is as hard as NP, the question remains whether such a connection is possible, and if it is, how should one go about establishing it.

The problem of basing cryptography on NP-hardness has played a central role since the beginnings of cryptography, and much research effort has been put into answering this question in the affirmative. A breakthrough was made in work by Ajtai [5], who showed that the existence of intractable problems in distributional NP follows from the assumption that there is no efficient algorithm that approximates the length of the shortest vector on a lattice in the worst case (within a factor of $n^{O(1)}$, where n is the dimension of the lattice). This is the first example of a problem in distributional NP whose hardness follows from a reasonable worst-case intractability assumption. In later works, Ajtai, Dwork, Micciancio, and Regev substantially extended Ajtai's original result, showing that (i) The existence of useful cryptographic objects, including one-way functions and public key encryption schemes, also follows from reasonable worst-case intractability assumptions and (ii) the worst-case intractability assumption used by Ajtai can be substantially weakened, giving the hope that further improvements could replace Ajtai's assumption with the strongest possible worst-case intractability assumption, namely NP $\not\subseteq$ BPP.

All known worst-case to average-case connections for NP are established by reductions, and all known reductions start from a problem that is known to reside inside NP $\cap$ coNP. One view of this situation is that membership in NP $\cap$ coNP does not reveal anything fundamental about the relation between worst-case and average-case complexity for NP, but is merely an artifact of the current reductions; improved reductions could go beyond this barrier, and eventually yield an equivalence between worst-case and average-case hardness for NP.

On the other hand, the results presented in this section, if liberally interpreted, seem to indicate the opposite: The mere existence of a worst-case to average-case reduction for NP often implies that

the problem one is reducing from is in NP $\cap$ coNP (or AM $\cap$ coAM, or NP/poly $\cap$ coNP/poly.) Moreover, the reason for this connection appears to be fairly universal: A worst-case to average-case reduction can be viewed as a proof system in which the verifier runs the reduction, and the prover simulates the average-case oracle. The difficulty is in forcing even a cheating prover to simulate the average-case oracle correctly; currently, it is known how to do this only under restrictive assumptions on the reduction (Theorems 7.5 and 7.6). However, further improvements may lead to the conclusion that this connection between worst-case to average-case reduction and constant-round proof systems is a universal one, and thus there is no hope of basing average-case complexity for NP on NP-hardness assumptions by means of a reduction.

8

Other Topics

The theory of average-case complexity for NP lacks the wealth of natural complete problems encountered in worst-case complexity. Yet, there are many natural distributional problems that are believed to be intractable on average.

One such problem is random kSAT, whose instances are generated by choosing clauses independently at random. In Section 8.1 we survey some of the known results about random kSAT, especially for $k = 3$. While random 3SAT is not known to be average-case complete, some versions of it are not known to have efficient errorless heuristics. An unusual result of Feige shows that the intractability of random 3SAT would have some interesting consequences in approximation complexity.

Another class of problems that are believed to be intractable on average is derived from lattice-based cryptography. The importance of these problems stems from the fact that they are the only known examples of problems in distributional NP that are hard according to a worst-case notion of hardness: If these problems were easy on average, then the corresponding problems on lattices, long believed to be hard, could be solved in the worst case. We survey some key results in Section 8.2.

8.1 The complexity of random kSAT

A widely investigated question in both statistics and the theory of computing is the tractability of random kCNF instances with respect to natural distributions. The most widely studied distribution on kCNF instances is the following: Given parameters $n > 0$ and $m_k(n) > 0$, choose at random $m_k(n)$ out of the $2^k\binom{n}{k}$ possible clauses of a kCNF on n boolean variables. An essentially equivalent model is to choose each of the possible $2^k\binom{n}{k}$ clauses independently with probability $m_k(n)/2^k\binom{n}{k}$.

By a counting argument, it follows that when $m_k(n)/n \geq 2^k \ln 2$, a random kCNF is almost always unsatisfiable as n grows large. Better analysis improves this upper bound by a small additive constant. Achlioptas and Peres [2], following Achlioptas and Moore [1], prove that when $m_k(n) < 2^k \ln 2 - k \ln 2/2 - c$ (for a constant c), then a random kCNF is almost always satisfiable. Their result is non-constructive, that is, they do not provide an efficient algorithm that finds satisfying assignments for a large fraction of such formulas.

For specific values of k, better lower and upper bounds are known. All known such lower bounds, except for the Achlioptas-Peres and Achlioptas-Moore results, are algorithmic. In particular, it is known that $3.51 < m_3(n)/n < 4.51$.

Friedgut [28] showed that for every $k \geq 2$, satisfiability of random kCNF exhibits a (possibly) non-uniform threshold. More precisely, for every $\varepsilon > 0$ and sufficiently large n there exists a value $c_k(n)$ such that a random kCNF is satisfiable with probability $1 - \varepsilon$ when $m_k(n)/n \leq (1 - \varepsilon)c_k(n)$, and with probability at most ε when $m_k(n)/n \geq (1 + \varepsilon)c_k(n)$. It is conjectured that the sequence $c_k(n)$ converges to a value c_k, known as the kSAT threshold, as $n \to \infty$. Experiments indicate for instance that $c_3(n) \to c_3 \approx 4.26$.

Assuming the existence of a threshold for kSAT, the existence of heuristic algorithms for random kSAT with respect to this family of distributions becomes trivial everywhere except possibly at the threshold.[1] However, the situation is different with respect to errorless

[1] In the literature on random kSAT, usually the error parameter of the average-case algorithm is implicitly fixed to $o(1)$ or n^{-c} for some fixed c. Not much is known for the case of algorithms with negligible error or heuristic schemes.

algorithms. Below the threshold, where most of the formulas are satisfiable, an errorless algorithm must certify most satisfiable formulas efficiently. In fact, since the lower bounds for $m_k(n)$ are algorithmic, we know that for every k there is an errorless algorithm for kSAT when $m_k(n)/n < a_k 2^k/k$, where the sequence a_k converges to some positive value. It is conjectured that algorithms for finding satisfying assignments on most kCNF instances exist all the way up to the kSAT threshold.

8.1.1 Refuting random CNF instances

Above the kSAT threshold, where most of the formulas are unsatisfiable, an errorless algorithm is required to refute most kCNF instances efficiently. A useful way of thinking of such a refutation algorithm is the following: The algorithm is given a kCNF instance φ and wants to distinguish between the case when φ is satisfiable and when φ is "typical" for the distribution on inputs. The algorithm can subject φ to any efficiently computable test that a random φ passes with high probability. If the instance φ does not pass these tests, the algorithm can output $\perp$. The challenge is to design a set of tests such that every φ that passes all the tests must be unsatisfiable, in which case the algorithm rejects φ.

When $m_k(n) > \Omega_k(n^{k-1})$, the following naive refutation algorithm works: Take a variable, say x_1, and consider all the clauses that contain it. Fixing x_1 to true yields a $(k-1)$CNF consisting of those $\Omega_k(n^{k-2})$ clauses that contain the literal $\overline{x}_1$, and this formula can be refuted recursively (the base case being a 2CNF, for which an efficient refutation algorithm exists). Repeat by fixing x_1 to false. (For an improved version of this approach, see [11].)

A more sophisticated approach for refuting random kCNF that handles smaller values of $m_k(n)$ was introduced by Goerdt and Krivelevich [29]. Their idea is to reduce kCNF instances to graphs (using a variant of Karp's reduction from 3SAT to maximum independent set) so that satisfiable formulas map to graphs with large independent sets, while the image of a random kCNF instance is unlikely to have a large independent set. Moreover, they show that for most graphs

derived from random kCNF, it is possible to efficiently certify that the graph does not have a large independent set via eigenvalue computations. Subsequent improvements of this argument yield refutation algorithms for random kCNF with $m_k(n) = \omega(n^{\lceil k/2 \rceil})$ [19]. For the case $k = 3$ there are better refutation algorithms, and the best known works for $m_3(n) = \omega(n^{3/2})$ [25]. This algorithm departs from previous work in that it does not reduce 3SAT to maximum independent set but uses a different reduction by Feige [23], which we describe in the next section.

Do refutation algorithms for random kCNF exist when $m_k(n)$ is above the satisfiability threshold $c_k n$, but below $n^{k/2}$? For the case of 3CNF, there is evidence suggesting that refuting random formulas may be hard for $m_3(n) < n^{3/2-\varepsilon}$ for every $\varepsilon > 0$. Ben-Sasson and Wigderson [13] (following [18]) show that for this range of parameters, most formulas require refutations *by resolution* of size $2^{\Omega(n^{\varepsilon/(1-\varepsilon)})}$. (The naive refutation algorithm above can be viewed as implementing a simple proof by resolution.) Recently, Feige and Ofek [26] showed that a different approach based on semi-definite programming that subsumes the algorithm of [25] also fails to certify unsatisfiability when $m_3(n) < n^{3/2}/\mathrm{poly}\log(n)$.

A very recent breakthrough of Feige, Kim, and Ofek [24] gives a *nondeterministic* refutation algorithm for $m_3(n) = \omega(n^{7/5})$, thus showing that random 3SAT with respect to this distribution is in $\mathrm{Avg}_{o(1)}\mathrm{coNP}$.[2]

8.1.2 Connection to hardness of approximation

Feige [23] conjectures that for every constant c, unsatisfiability of random 3CNF is hard to certify (within negligible error) whenever $m_3(n) < cn$. In particular, Feige's conjecture implies that $(\mathrm{NP}, \mathrm{PSAMP}) \not\subseteq \mathrm{Avg}_{\mathrm{neg}}\mathrm{P}$, but there is no evidence as to whether random 3SAT with parameter $m_3(n) < cn$ is complete for the class $(\mathrm{NP}, \mathrm{PSAMP})$.

Instead of pursuing connections with average-case complexity, Feige views his conjecture as a strengthening of the famous result by Håstad [38] about the inapproximability of 3SAT in the worst case. Indeed, Håstad shows that assuming $\mathrm{P} \neq \mathrm{NP}$, it is hard to distinguish between satisfiable 3CNF instances and 3CNF instances where

[2] This class is defined in a way analogous to $\mathrm{Avg}_{\delta}\mathrm{P}$; see Section 2.2.

no more than a $\frac{7}{8} + \varepsilon$ fraction of the clauses can be satisfied. The class of instances on which no more than $\frac{7}{8} + \varepsilon$ fraction of the clauses can be satisfied in particular includes most random 3CNF instances with cn clauses for sufficiently large c. Feige's conjecture says that even if we restrict ourselves to these random instances, the distinguishing problem remains intractable. As several inapproximability results assuming $\mathrm{P} \neq \mathrm{NP}$ follow by reduction from the hardness of approximating 3SAT, it can be hoped that Feige's stronger conjecture may yield new or stronger conclusions.

The main technical result of Feige is the following theorem. For notation purposes, given a 3CNF φ and an assignment a, let $\mu_i(\varphi, a)$ denote the fraction of clauses in φ, where a satisfies exactly i literals, for $0 \leq i \leq 3$.

Theorem 8.1. (Feige) For every $\varepsilon > 0$ there exists an algorithm A that for all sufficiently large c, has the following properties:

(1) A accepts all but a negligible fraction of random 3CNF on n variables and cn clauses.
(2) For sufficiently large n, if φ is a satisfiable 3CNF with n variables and cn clauses and A accepts φ, then for every satisfying assignment a of φ, it holds that $\mu_1(\varphi, a) = 3/4 \pm \varepsilon$, $\mu_2(\varphi, a) < \varepsilon$, and $\mu_3(\varphi, a) = 1/4 \pm \varepsilon$.

Observe that, in contrast, for most random 3CNF φ and every assignment a, we have that $\mu_1(\varphi, a) = \mu_2(\varphi, a) = \frac{3}{8} \pm \varepsilon$ and $\mu_0(\varphi, a) = \mu_3(\varphi, a) = \frac{1}{8} \pm \varepsilon$.

Assuming the conjecture, the theorem, for instance, implies the following: For a 3CNF φ with n variables and cn clauses, it is hard to distinguish between the following cases:

(1) There exists an assignment for φ that satisfies *all* literals in a $\frac{1}{4} - \varepsilon$ fraction of clauses
(2) No assignment for φ satisfies all literals in more than a $\frac{1}{8} + \varepsilon$ fraction of clauses.

This hardness of approximation result is not known to follow from $P \neq NP$. Feige shows that hardness of approximation results for balanced bipartite clique, min bisection, dense subgraph, and the 2-catalog problem follow from it[3] via combinatorial reductions.

8.2 The complexity of lattice problems

Discrete lattices in $\mathbb{R}^n$ provide examples of problems in NP that are believed to be intractable in the worst case and which worst-case to average-case reduce to certain distributional problems in $(NP, PSAMP)$. Some of these reductions yield stronger objects such as one-way functions, collision-resistant hash functions, and public key cryptosystems.

The lattice problems in question are all *promise* problems [21, 32]. Instead of attempting to list all their variants and the connections between them, for illustration we focus on the shortest vector problem. (Other lattice problems exhibit similar behavior. For a more general treatment, see [58] and [59].) A lattice $\mathcal{L}$ in $\mathbb{R}^n$ is represented by specifying a basis of n vectors for it (all vectors have $\text{poly}(n)$ size descriptions).

The shortest vector problem $\text{SVP}_{\gamma(n)}$: The instances are pairs $(\mathcal{L}, d)$, where $\mathcal{L}$ is a lattice in $\mathbb{R}^n$ and d is a number. In yes instances, there exists a vector $\mathbf{v}$ in $\mathcal{L}$ of length at most d.[4] In no instances, every vector in $\mathcal{L}$ has length at least $\gamma(n)d$.

This problem is in NP (for $\gamma(n) \geq 1$.) The following seemingly easier variant also turns out to be useful.

The unique shortest vector problem $\text{uSVP}_{\gamma(n)}$: This is the same as $\text{SVP}_{\gamma(n)}$, except that in yes instances we require that every vector in $\mathcal{L}$ whose length is at most $\gamma(n)d$ be parallel to the shortest vector v.

We stress that we are interested in the *worst-case* hardness of these problems as the dimension of the lattice n grows. The best-known polynomial-time approximation algorithm for the shortest vector problem, due to Ajtai, Kumar, and Sivakumar [8], solves $\text{SVP}_{\gamma(n)}$ for $\gamma(n) = 2^{\Theta(n \log\log n / \log n)}$ (previous algorithms of Lenstra, Lenstra,

[3] To be precise, Feige proves and needs a slightly more general result.

[4] To be specific we measure length in the ℓ_2 norm. The problem is no easier for other ℓ_p norms, see [66].

and Lovász [52] and Schnorr [67] achieve somewhat worse approximation factors). For polynomial approximation factors $\gamma(n) = \text{poly}(n)$, the best-known algorithms run in time $2^{\Theta(n)}$ [8, 50].

In a seminal article, Ajtai [5] showed that assuming $\text{SVP}_{O(n^c)}$ is intractable for some fixed $c > 0$ there exist one-way functions. He constructs a family of functions $\{f_n\}$ for which there exists a worst-case to average-case reduction from $\text{SVP}_{O(n^c)}$ to inverting $\{f_n\}$. Later, Ajtai and Dwork [7] showed that public key encryption exists assuming $\text{uSVP}_{O(n^c)}$ is intractable for some fixed $c > 0$. The parameter c has been improved since the original constructions, and it is known that

(1) One-way functions and collision-resistant hash functions exist assuming $\text{SVP}_{\tilde{O}(n)}$ is intractable [59].
(2) Public key encryption exists assuming $\text{uSVP}_{\tilde{O}(n^{1.5})}$ is intractable [63].
(3) Public key encryption exists assuming $\text{SVP}_{\tilde{O}(n^{1.5})}$ is intractable by quantum algorithms [64].

A short, self-contained outline of a basic worst-case to average-case reduction from uSVP can be found in a tutorial of Regev [65].

These results greatly motivate the study of hardness of lattice problems: For instance, if it were true that $\text{SVP}_{n^{1.5+\varepsilon}}$ is NP-hard for some $\varepsilon > 0$, it would follow that one-way functions exist (and in particular $(\text{NP}, \text{PSAMP}) \not\subseteq \text{HeurBPP}$) assuming only $\text{NP} \not\subseteq \text{BPP}$.

However, the best hardness results known for the shortest vector problem fall short of what is necessary for the current worst-case to average-case reductions. Micciancio [56] (following Ajtai [6]) showed that $\text{SVP}_{\gamma(n)}$ where $\gamma(n) = \sqrt{2} - \varepsilon$ is NP-hard under randomized polynomial-time reductions for every $\varepsilon > 0$. More recently, Khot [48] improved the hardness to $\gamma(n) = 2^{(\log n)^{1/2-\varepsilon}}$ for every $\varepsilon > 0$, but his reduction runs in randomized quasipolynomial time.

On the other hand, Goldreich and Goldwasser [30] showed that $\text{SVP}_{\gamma(n)} \in \text{coAM}$ for $\gamma(n) = \Omega(\sqrt{n/\log n})$ and Aharonov and Regev [4] showed that $\text{SVP}_{\gamma(n)} \in \text{coNP}$ for $\gamma(n) = \Omega(\sqrt{n})$. This can be taken as evidence that $\text{SVP}_{\gamma(n)}$ is not NP-hard when $\gamma(n)$ exceeds $\sqrt{n}$, but one must be careful because $\text{SVP}_{\gamma(n)}$ is a promise problem, not a language. While it is true that assuming $\text{NP} \neq \text{coNP}$, languages in $\text{NP} \cap \text{coNP}$

cannot be NP-hard, this conclusion fails in general for promise problems: Even, Selman, and Yacobi [21] give an example of a promise problem that is NP-hard yet resides in NP $\cap$ coNP.

It is interesting to observe that the one-way functions constructed by Ajtai [5] and Micciancio and Regev [59] are size-approximable (in fact, almost regular), so by Theorem 7.4 in the best case the hardness of these functions can be based on problems in AM $\cap$ coAM.

A

Samplable Ensembles Versus Samplable Distributions

In the work of Ben-David *et al.* [12] that explains and extends Levin's original definitions [53], a distribution over $\{0,1\}^*$ is considered samplable if it is generated by a randomized algorithm S that runs in time polynomial in the length of its *output*.

Working with ensembles of samplable distributions instead of a single samplable distribution does not incur any loss of generality: In fact, for every samplable distribution $\mathcal{D}$ there exists a samplable ensemble $\{D_n\}$ such that A is a heuristic scheme with respect to $\mathcal{D}$ if and only if some algorithm A' (a slight modification of A) is a heuristic scheme with respect to $\{D_n\}$. (The equivalence preserves the errorless property of heuristic schemes.)

To sketch the proof, let X_n be the set of all $x \in \{0,1\}^*$ such that the sampler S for $\mathcal{D}$ outputs x in n or fewer steps. Let D_n be the distribution $\mathcal{D}$ conditioned on the event $x \in X_n$, so that for every $x \in X_n$, $D_n(x) = \mathcal{D}(x)/\mathcal{D}(X_n)$. Let n_0 be the smallest n for which $\mathcal{D}(X_n) \geq \frac{1}{2}$. The ensemble $\{D_n\}$ is samplable,[1] the support of D_n is contained in $\{0,1\}^{\leq n}$, and $\mathcal{D}(X_n) = 1 - o_n(1)$.

[1] When $n \geq n_0$, run S for n steps repeatedly until a sample is produced; for smaller n, the distribution D_n can be hard coded in the sampler. This sampler runs in *expected*

Given an algorithm A that is good on average for $\mathcal{D}$, we define

$$A'(x;n,\delta) = \begin{cases} A(x;\delta/2), & \text{if } n \geq n_0, \\ L(x), & \text{otherwise.} \end{cases}$$

For $n < n_0$, the distribution D_n contains strings of length at most n_0, and the answers for these inputs are hardcoded into A'. For $n \geq n_0$, we have

$$\begin{aligned} \mathbf{Pr}_{x\sim D_n}[A'(x;n,\delta) \neq L(x)] &\leq \mathbf{Pr}_{x\sim\mathcal{D}}[A'(x;n,\delta) \neq L(x)]/\mathcal{D}(X_n) \\ &\leq \mathbf{Pr}_{x\sim\mathcal{D}}[A(x;\delta/2) = \perp]/\tfrac{1}{2} \leq \delta. \end{aligned}$$

Conversely, given an algorithm A' that is good on average for $\{D_n\}$, we define

$$A(x;\delta) = A'(x;p(|x|),\delta/2|x|^2),$$

where $p(n)$ is an upper bound on the time it takes S to output a string of length n. We have

$$\begin{aligned} &\mathbf{Pr}_{x\sim\mathcal{D}}[A(x;\delta) \neq L(x)] \\ &\quad = \mathbf{Pr}_{x\sim\mathcal{D}}[A'(x;p(|x|),\delta/2|x|^2) \neq L(x)] \\ &\quad = \sum_{n=0}^{\infty} \mathbf{Pr}_{x\sim\mathcal{D}}[A'(x;p(n),\delta/2n^2) \neq L(x) \text{ and } |x| = n] \\ &\quad \leq \sum_{n=0}^{\infty} \mathbf{Pr}_{x\sim\mathcal{D}}[A'(x;p(n),\delta/2n^2) \neq L(x) \text{ and } S \to x \text{ in } p(n) \text{ steps}] \\ &\quad \leq \sum_{n=0}^{\infty} \mathbf{Pr}_{x\sim D_{p(n)}}[A'(x;p(n),\delta/2n^2) \neq L(x)] \\ &\quad \leq \sum_{n=0}^{\infty} \delta/2n^2 < \delta. \end{aligned}$$

---

polynomial time, so D_n does not in fact satisfy the definition on perfect samplability; however, it is within statistical distance $2^{-\mathrm{poly}(n)}$ of a samplable distribution, and we will ignore the distinction.

Acknowledgments

We thank Scott Aaronson, Jonathan Katz, Chris Moore and the anonymous referee for their helpful comments.

References

[1] D. Achlioptas and C. Moore, "The asymptotic order of the k-SAT threshold," in *Proceedings of the 43rd IEEE Symposium on Foundations of Computer Science*, pp. 779–788, 2002.

[2] D. Achlioptas and Y. Peres, "The threshold for random k-SAT is $2^k \log 2 - O(k)$," *Journal of the AMS*, vol. 17, no. 4, pp. 947–973, 2004.

[3] L. Adleman, "Two theorems on random polynomial time," in *Proceedings of the 19th IEEE Symposium on Foundations of Computer Science*, pp. 75–83, 1978.

[4] D. Aharonov and O. Regev, "Lattice Problems in NP∩coNP," *Journal of the ACM*, vol. 52, no. 5, pp. 749–765, Preliminary version in Proceedings of FOCS 2004, 2005.

[5] M. Ajtai, "Generating hard instances of lattice problems," in *Proceedings of the 28th ACM Symposium on Theory of Computing*, pp. 99–108, 1996.

[6] M. Ajtai, "The Shortest Vector Problem in ℓ_2 is NP-hard for Randomized Reductions," in *Proceedings of the 30th ACM Symposium on Theory of Computing*, pp. 10–19, 1998.

[7] M. Ajtai and C. Dwork, "A Public-Key Cryptosystem with Worst-Case/Average-Case Equivalence," in *Proceedings of the 29th ACM Symposium on Theory of Computing*, pp. 284–293, 1997.

[8] M. Ajtai, R. Kumar, and D. Sivakumar, "A sieve algorithm for the shortest lattice vector problem," in *Proceedings of the 33rd ACM Symposium on Theory of Computing*, pp. 601–610, 2001.

[9] A. Akavia, O. Goldreich, S. Goldwasser, and D. Moshkovitz, "On basing one-way functions on NP-hardness," in *Proceedings of the 38th ACM Symposium on Theory of Computing*, 2006.

[10] L. Babai, L. Fortnow, N. Nisan, and A. Wigderson, "BPP has subexponential time simulations unless EXPTIME has publishable proofs," *Computational Complexity*, vol. 3, no. 4, pp. 307–318, 1993.

[11] P. Beame, R. Karp, T. Pitassi, and M. Saks, "On the complexity of unsatisfiability proofs for random k-CNF formulas," in *Proceedings of the 30th ACM Symposium on Theory of Computing*, 1998.

[12] S. Ben-David, B. Chor, O. Goldreich, and M. Luby, "On the theory of average case complexity," *Journal of Computer and System Sciences*, vol. 44, no. 2, pp. 193–219, 1992.

[13] E. Ben-Sasson and A. Wigderson, "Short proofs are narrow: Resolution made simple," *Journal of the ACM*, vol. 48, no. 2, 2001.

[14] M. Blum and S. Micali, "How to generate cryptographically strong sequences of pseudorandom bits," *SIAM Journal on Computing*, vol. 13, no. 4, pp. 850–864, Preliminary version in *Proceedings of FOCS'82*, 1984.

[15] A. Bogdanov and L. Trevisan, "On wosrt-case to average-case reductions for NP problems," in *Proceedings of the 44th IEEE Symposium on Foundations of Computer Science*, pp. 308–317, 2003.

[16] D. Boneh and R. J. Lipton, "Amplification of weak learning under the uniform distribution," in *Proceedings of the 6th ACM Conference on Computational Learning Theory*, pp. 347–351, 1993.

[17] G. Brassard, "Relativized cryptography," in *Proceedings of the 20th IEEE Symposium on Foundations of Computer Science*, pp. 383–391, 1979.

[18] V. Chvatal and E. Szemeredi, "Many hard examples for resolution," *Journal of the ACM*, vol. 35, no. 4, pp. 759–768, 1998.

[19] A. Coja-Oghlan, A. Goerdt, A. Lanka, and F. Schdlich, "Certifying unsatisfiability of random 2k-SAT formulas using approximation techniques," in *Proceedings of 14th Symposium on Foundations of Computation Theory*, pp. 15–26, LNCS 2751, 2003.

[20] W. Diffie and M. E. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, 1976.

[21] S. Even, A. Selman, and Y. Yacobi, "The complexity of promise problems with applications to public-key cryptography," *Information and Computation*, vol. 61, no. 2, pp. 159–173, 1984.

[22] S. Even and Y. Yacobi, "Cryptography and NP-completeness," in *Proceedings of the 7th International Colloquium on Automata, Languages and Programming*, pp. 195–207, Springer-Verlag, 1980.

[23] U. Feige, "Relations between average case complexity and approximation complexity," in *Proceedings of the 34th ACM Symposium on Theory of Computing*, pp. 534–543, 2002.

[24] U. Feige, J. Kim, and E. Ofek, "Witnesses for non-satisfiability of dense random 3CNF formulas," in *Proceedings of the 47th IEEE Symposium on Foundations of Computer Science*, 2006. To appear.

[25] U. Feige and E. Ofek, "Easily refutable subformulas of random 3CNF formulas," in *Proceedings of the 31st International Colloquium on Automata, Languages and Programming*, pp. 519–530, 2004.

[26] U. Feige and E. Ofek, "Random 3CNF formulas elude the Lovasz theta function," Tech. Rep. TR06-043, Electronic Colloquium on Computational Complexity, 2006.

[27] J. Feigenbaum and L. Fortnow, "Random-self-reducibility of complete sets," *SIAM Journal on Computing*, vol. 22, pp. 994–1005, 1993.

[28] E. Friedgut, "Necessary and sufficient conditions for sharp thresholds of graph properties and the k-SAT problem," *Journal of the AMS*, vol. 12, pp. 1017–1054, 1999.

[29] A. Goerdt and M. Krivelevich, "Efficient recognition of random unsatisfiable k-SAT instances by spectral methods," in *Proceedings of the 18th Symposium on Theoretical Aspects of Computer Science*, pp. 294–304, 2001.

[30] O. Goldreich and S. Goldwasser, "On the limits of non-approximability of lattice problems," in *Proceedings of the 30th ACM Symposium on Theory of Computing*, pp. 1–9, 1998.

[31] O. Goldreich, *The Foundations of Cryptography - Volume 1*. Cambridge University Press, 2001.

[32] O. Goldreich, "On promise problems (a survey in memory of Shimon Even [1935–2004])," Tech. Rep. TR05-018, Electronic Colloquium on Computational Complexity, 2005.

[33] O. Goldreich and S. Goldwasser, "On the possibility of basing Cryptography on the assumption that $P \neq NP$," 1998. Unpublished manuscript.

[34] O. Goldreich, N. Nisan, and A. Wigderson, "On Yao's XOR Lemma," Tech. Rep. TR95-50, Electronic Colloquium on Computational Complexity, 1995.

[35] S. Goldwasser and S. Micali, "Probabilistic encryption," *Journal of Computer and System Sciences*, vol. 28, no. 2, pp. 270–299, Preliminary Version in *Proceedings of STOC'82*, 1984.

[36] D. Gutfreund, R. Shaltiel, and A. Ta-Shma, "If NP languages are hard on the worst-case then it is easy to find their hard instances," in *Proceedings of the 20th IEEE Conference on Computational Complexity*, 2005.

[37] D. Gutfreund and A. Ta-Shma, "New connections between derandomization, worst-case complexity and average-case complexity," Tech. Rep. TR06-108, Electronic Colloquium on Computational Complexity, 2006.

[38] J. Håstad, "Some optimal inapproximability results," *Journal of the ACM*, vol. 48, no. 4, pp. 798–859, 2001.

[39] J. Håstad, R. Impagliazzo, L. Levin, and M. Luby, "A pseudorandom generator from any one-way function," *SIAM Journal on Computing*, vol. 28, no. 4, pp. 1364–1396, 1999.

[40] A. Healy, S. Vadhan, and E. Viola, "Using nondeterminism to amplify hardness," in *Proceedings of the 36th ACM Symposium on Theory of Computing*, pp. 192–201, 2004.

[41] R. Impagliazzo, "Hard-core distributions for somewhat hard problems," in *Proceedings of the 36th IEEE Symposium on Foundations of Computer Science*, pp. 538–545, 1995.

[42] R. Impagliazzo and L. Levin, "No Better Ways to Generate Hard NP Instances than Picking Uniformly at Random," in *Proceedings of the 31st IEEE Symposium on Foundations of Computer Science*, pp. 812–821, 1990.

[43] R. Impagliazzo and M. Luby, "One-way Functions are Essential for Complexity Based Cryptography," in *Proceedings of the 30th IEEE Symposium on Foundations of Computer Science*, pp. 230–235, 1989.

[44] R. Impagliazzo and A. Wigderson, "$P = BPP$ unless E has sub-exponential circuits," in *Proceedings of the 29th ACM Symposium on Theory of Computing*, pp. 220–229, 1997.

[45] V. Kabanets, "Derandomization: A brief overview," *Bulletin of the European Association for Theoretical Computer Science*, vol. 76, pp. 88–103, 2002.

[46] R. Karp, "Probabilistic Analysis of Partitioning Algorithms for the Traveling-Salesman Problem in the Plane," *Mathematics of Operations Research*, vol. 2, no. 3, pp. 209–224, 1977.

[47] R. Karp, J. Lenstra, C. McDiarmid, and A. R. Kan, "Probabilistic analysis," in *Combinatorial Optimization: An Annotated Bibliography*, (M. O'hEigeartaigh, J. Lenstra, and A. R. Kan, eds.), pp. 52–88, Wiley, 1985.

[48] S. Khot, "Hardness of approximating the shortest vector problem in lattices," 2004. Manuscript.

[49] D. Knuth, *The Art of Computer Programming*. Vol. 3, Addison-Wesley, 1973.

[50] R. Kumar and D. Sivakumar, "On polynomial-factor approximations to the shortest lattice vector length," *SIAM Journal on Discrete Mathematics*, vol. 16, no. 3, pp. 422–425, Preliminary version in Proceedings of SODA 2001, 2003.

[51] A. Lempel, "Cryptography in transition," *Computing Surveys*, vol. 11, no. 4, pp. 215–220, 1979.

[52] A. Lenstra, H. Lenstra, and L. Lovasz, "Factoring polynomials with rational coefficients," *Mathematische Annalen*, vol. 261, pp. 515–534, 1982.

[53] L. Levin, "Average case complete problems," *SIAM Journal on Computing*, vol. 15, no. 1, pp. 285–286, 1986.

[54] L. Levin, "One-Way Functions and Pseudorandom Generators," *Combinatorica*, vol. 7, no. 4, pp. 357–363, 1987.

[55] M. Li and P. M. B. Vitányi, "Average case complexity under the universal distribution equals worst-case complexity," *IPL*, vol. 42, no. 3, pp. 145–149, 1992.

[56] D. Micciancio, "The shortest vector problem is NP-hard to approximate to within some constant," *SIAM Journal on Computing*, vol. 30, no. 6, pp. 2008–2035, 2001.

[57] D. Micciancio, "Almost perfect lattices, the covering radius problem, and applications to Ajtai's connection factor," *SIAM Journal on Computing*, vol. 34, no. 1, pp. 118–169, 2004.

[58] D. Micciancio and S. Goldwasser, *Complexity of Lattice Problems*. Norwell, MA, USA: Kluwer Academic Publishers, 2002.

[59] D. Micciancio and O. Regev, "Worst-case to average-case reductions based on gaussian measure," in *Proceedings of the 45th IEEE Symposium on Foundations of Computer Science*, pp. 372–381, 2004.

[60] N. Nisan and A. Wigderson, "Hardness vs randomness," *Journal of Computer and System Sciences*, vol. 49, pp. 149–167, Preliminary version in *Proc. of FOCS'88*, 1994.

[61] R. O'Donnell, "Hardness amplification within NP," in *Proceedings of the 34th ACM Symposium on Theory of Computing*, pp. 751–760, 2002.

[62] R. Ostrovsky, "One-way functions, hard on average problems and statistical zero-knowledge proofs," in *STRUCTURES91*, pp. 51–59, 1991.

[63] O. Regev, "New lattice based cryptographic constructions," in *Proceedings of the 35th ACM Symposium on Theory of Computing*, pp. 407–416, 2003.

[64] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," in *Proceedings of the 37th ACM Symposium on Theory of Computing*, pp. 84–93, 2005.

[65] O. Regev, "Lattice-based cryptography," in *Advances in Cryptology (CRYPTO)*, pp. 131–141, 2006.

[66] O. Regev and R. Rosen, "Lattice problems and norm embeddings," in *Proceedings of the 38th ACM Symposium on Theory of Computing*, pp. 447–456, 2006.

[67] C. Schnorr, "A hierarchy of polynomial time lattice basis reduction algorithms," *Theoretical Computer Science*, vol. 53, pp. 201–224, 1987.

[68] A. Shamir, "On the cryptocomplexity of knapsack systems," in *Proceedings of the 11th ACM Symposium on Theory of Computing*, pp. 118–129, 1979.

[69] M. Sudan, L. Trevisan, and S. Vadhan, "Pseudorandom generators without the XOR Lemma," *Journal of Computer and System Sciences*, vol. 62, no. 2, pp. 236–266, 2001.

[70] L. Trevisan, "List-decoding using the XOR Lemma," in *Proceedings of the 44th IEEE Symposium on Foundations of Computer Science*, pp. 126–135, 2003.

[71] L. Trevisan, "Some Applications of coding theory in computational complexity," *Quaderni di Matematica*, vol. 13, pp. 347–424, arXiv:cs.CC/0409044, 2004.

[72] L. Trevisan, "On uniform amplification of hardness in NP," in *Proceedings of the 37th ACM Symposium on Theory of Computing*, pp. 31–38, 2005.

[73] L. G. Valiant and V. V. Vazirani, "NP is as easy as detecting unique solutions," *Theoretical Computer Science*, vol. 47, pp. 85–93, 1986.

[74] E. Viola, "The Complexity of constructing pseudorandom generators from hard functions," *Computational Complexity*, vol. 13, no. 3-4, pp. 147–188, 2004.

[75] E. Viola, "On Constructing Parallel Pseudorandom Generators from One-Way Functions," in *Proceedings of the 20th IEEE Conference on Computational Complexity*, 2005.

[76] A. C. Yao, "Theory and applications of trapdoor functions," in *Proceedings of the 23th IEEE Symposium on Foundations of Computer Science*, pp. 80–91, 1982.