
Average-Case Complexity

Average-Case Complexity

Andrej Bogdanov

DIMACS – Rutgers University
adib@dimacs.rutgers.edu

Luca Trevisan

UC Berkeley
luca@eecs.berkeley.edu

now

the essence of **know**ledge

Boston – Delft

Foundations and Trends[®] in Theoretical Computer Science

Published, sold and distributed by:

now Publishers Inc.
PO Box 1024
Hanover, MA 02339
USA
Tel. +1-781-985-4510
www.nowpublishers.com
sales@nowpublishers.com

Outside North America:

now Publishers Inc.
PO Box 179
2600 AD Delft
The Netherlands
Tel. +31-6-51115274

Library of Congress Control Number: 2006935853

The preferred citation for this publication is A. Bogdanov and L. Trevisan, Average-Case Complexity, Foundations and Trends[®] in Theoretical Computer Science, vol 2, no 1, pp 1–106, 2006

Printed on acid-free paper

ISBN: 1-933019-49-2

© 2006 A. Bogdanov and L. Trevisan

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, mechanical, photocopying, recording or otherwise, without prior written permission of the publishers.

Photocopying. In the USA: This journal is registered at the Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923. Authorization to photocopy items for internal or personal use, or the internal or personal use of specific clients, is granted by now Publishers Inc for users registered with the Copyright Clearance Center (CCC). The 'services' for users can be found on the internet at: www.copyright.com

For those organizations that have been granted a photocopy license, a separate system of payment has been arranged. Authorization does not extend to other kinds of copying, such as that for general distribution, for advertising or promotional purposes, for creating new collective works, or for resale. In the rest of the world: Permission to photocopy must be obtained from the copyright owner. Please apply to now Publishers Inc., PO Box 1024, Hanover, MA 02339, USA; Tel. +1 781 871 0245; www.nowpublishers.com; sales@nowpublishers.com

now Publishers Inc. has an exclusive license to publish this material worldwide. Permission to use this content must be obtained from the copyright license holder. Please apply to now Publishers, PO Box 179, 2600 AD Delft, The Netherlands, www.nowpublishers.com; e-mail: sales@nowpublishers.com

**Foundations and Trends[®] in
Theoretical Computer Science**
Volume 2 Issue 1, 2006
Editorial Board

Editor-in-Chief:

Madhu Sudan

*Department of CS and EE
MIT, Stata Center, Room G640
32 Vassar Street, Cambridge
Massachusetts 02139,
USA
madhu@mit.edu*

Editors

Bernard Chazelle (Princeton)
Oded Goldreich (Weizmann Inst.)
Shafi Goldwasser (MIT and Weizmann Inst.)
Jon Kleinberg (Cornell University)
László Lovász (Microsoft Research)
Christos Papadimitriou (UC. Berkeley)
Prabhakar Raghavan (Yahoo!Research)
Peter Shor (MIT)
Madhu Sudan (MIT)
Éva Tardos (Cornell University)
Avi Wigderson (IAS)

Editorial Scope

Foundations and Trends[®] in Theoretical Computer Science will publish survey and tutorial articles in the following topics:

- Algorithmic game theory
- Computational algebra
- Computational aspects of combinatorics and graph theory
- Computational aspects of communication
- Computational biology
- Computational complexity
- Computational geometry
- Computational learning
- Computational Models and Complexity
- Computational Number Theory
- Cryptography and information security
- Data structures
- Database theory
- Design and analysis of algorithms
- Distributed computing
- Information retrieval
- Operations Research
- Parallel algorithms
- Quantum Computation
- Randomness in Computation

Information for Librarians

Foundations and Trends[®] in Theoretical Computer Science, 2006, Volume 2, 4 issues. ISSN paper version 1551-305X. ISSN online version 1551-3068. Also available as a combined paper and online subscription.

Average-Case Complexity

Andrej Bogdanov¹ and Luca Trevisan²

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 $P \neq NP$ 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.

Contents

1	Introduction	1
1.1	Roadmap	3
1.2	A historical overview	9
2	Definitions of “Efficient on Average”	17
2.1	Distribution over inputs	17
2.2	Heuristic and errorless algorithms	20
2.3	Non-uniform and randomized heuristics	26
2.4	Representing inputs	30
2.5	A distribution for which worst case and average case are equivalent	31
3	A Complete Problem for Computable Ensembles	35
3.1	Reductions between distributional problems	35
3.2	The completeness result	37
3.3	Some observations	39
4	Decision Versus Search and One-Way Functions	43
4.1	Search algorithms	44
4.2	Reducing search to decision	46
4.3	Average-case complexity and one-way functions	49

5	Samplable Ensembles	51
5.1	The compressibility perspective	52
5.2	The invertibility perspective	61
6	Hardness Amplification	65
6.1	Yao's XOR Lemma	65
6.2	O'Donnell's approach	68
7	Worst-Case Versus Average-Case and Cryptography	73
7.1	Worst-case to average-case reductions	74
7.2	Permutations and range-computable functions	78
7.3	General one-way functions and average-case hard languages	82
7.4	Public key encryption	89
7.5	Perspective: Is distributional NP as hard as NP?	92
8	Other Topics	95
8.1	The complexity of random k SAT	96
8.2	The complexity of lattice problems	100
A	Samplable Ensembles Versus Samplable Distributions	103
	Acknowledgements	105
	References	107

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 $f : \{0,1\}^* \rightarrow \{0,1\}^*$ 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.

2 Introduction

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 $P \neq NP$ 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 $P = NP$, 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 inputs²; 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 $(\text{BH}, \mathcal{U}^{\text{BH}})$, where $\mathcal{U}^{\text{BH}}$ 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 $(\text{BH}, \mathcal{U}^{\text{BH}})$ 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 $(\text{BH}, \mathcal{U}^{\text{BH}})$ 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 $(\text{BH}, \mathcal{U}^{\text{BH}})$ 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 $(\text{BH}, \mathcal{U}^{\text{BH}})$, 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 $\text{P} = \text{NP}$.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

⁴This 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 $(\text{BH}, \mathcal{U}^{\text{BH}})$ 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 $x \rightarrow (r, C(x, r))$ 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 $P \neq \text{NP}$ 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). Gutfreund 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 mistakes⁵ 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].

⁵Or *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 k SAT 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 k SAT independently. The tractability of this problem appears to depend heavily on the number of clauses $m_k(n)$. While it is believed that random k SAT 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 [35,14,76] 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 $(\text{NP}, \text{PCOMP})$ 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 $(\text{NP}, \text{PSAMP})$ 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 $(\text{NP}, \text{PSAMP})$.

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 $\text{NP} = \text{coNP}$ (or $\text{AM} = \text{coAM}$,

if one allows randomized reductions, or $\text{NP}/\text{poly} = \text{coNP}/\text{poly}$, 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 $\text{P} = \text{NP}$. 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 $(\text{NP}, \text{PSAMP})$,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 vacuously.

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 $(\text{NP}, \text{PSAMP})$ 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 $(\text{NP}, \text{PCOMP})$ 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

⁸This 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 $\text{NP} \cap \text{coNP}$. 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 $\text{NP/poly} = \text{coNP/poly}$. Akavia *et al.* [9] prove that one-way functions cannot be based on NP-complete problems via non-adaptive reductions unless $\text{AM} = \text{coAM}$ (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 to 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 $x \in \{0,1\}^*$, $\Pr[A(n) = x] = D_n(x)$.
-

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 $P = P^{\#P}$. In fact, $P\text{COMP} = P\text{SAMP}$ if and only if $P = P^{\#P}$.

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 $(NP, P\text{SAMP})$, $(NP, P\text{COMP})$, and $(NP, \mathcal{U})$.

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 :

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

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

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

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

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

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

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

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

□

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 $\delta > 0$,

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

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 ,

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

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 $A(x; n, 1/8), \dots, A(x; n, 2^{-k}), \dots$ 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. $\square$

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 $\delta : \mathbb{N} \rightarrow \mathbb{R}^+$. 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 , $\Pr_{x \sim D_n}[A(x; n) = \perp] \leq \delta(n)$.

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

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

We use $\text{Avg}_{\text{neg}} \text{P}$ to denote the union of all classes $\text{Avg}_\delta \text{P}$, 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 $\delta : \mathbb{N} \rightarrow \mathbb{R}^+$. We say that an algorithm A is a *heuristic algorithm* for $(L, \mathcal{D})$ with *error probability* at most δ if for all $n > 0$,

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

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

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

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 $\text{Heur}_{\text{neg}} \text{P}$ to denote the union of all classes $\text{Heur}_\delta \text{P}$, 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, $\text{AvgC} \subseteq \text{HeurC}$ and $\text{Avg}_\delta \text{C} \subseteq \text{Heur}_\delta \text{C}$ 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 $(\text{NP}, \text{PSAMP})$ to $(\text{NP}, \mathcal{U})$ 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 $s : \mathbb{N} \rightarrow \mathbb{N}$, 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) $\Pr_{x \sim D_n}[C(x) \neq L(x)] \leq \delta(n)$.

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 $\perp$.