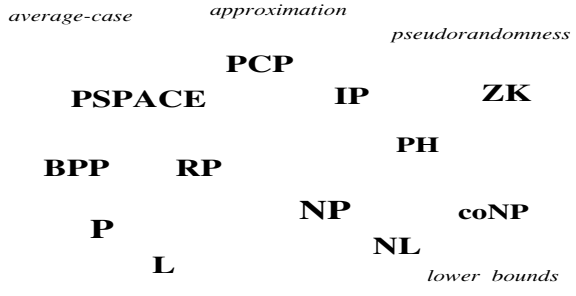
CHAPTER ONE

Introduction and Preliminaries

When you set out on your journey to Ithaca,
pray that the road is long,
full of adventure, full of knowledge.
K. P. Cavafy, "Ithaca"

The current chapter consists of two parts. The first part provides a high-level introduction to (computational) Complexity Theory. This introduction is much more detailed than the laconic statements made in the preface, but is quite sparse when compared to the richness of the field. In addition, the introduction contains several important comments regarding the contents, approach, and conventions of the current book.



The second part of this chapter provides the necessary preliminaries to the rest of the book. It includes a discussion of computational tasks and computational models, as well as natural complexity measures associated with the latter. More specifically, this part recalls the basic notions and results of computability theory (including the definition of Turing machines, some undecidability results, the notion of universal machines, and the definition of oracle machines). In addition, this part presents the basic notions underlying non-uniform models of computation (like Boolean circuits).

1.1. Introduction

This introduction consists of two parts: The first part refers to the area itself, whereas the second part refers to the current book. The first part provides a brief overview of Complexity Theory (Section 1.1.1) as well as some reflections about its characteristics (Section 1.1.2). The second part describes the contents of this book (Section 1.1.3), the

considerations underlying the choice of topics as well as the way they are presented (Section 1.1.4), and various notations and conventions (Section 1.1.5).

1.1.1. A Brief Overview of Complexity Theory

Out of the tough came forth sweetness[1]
Judges, 14:14

The following brief overview is intended to give a flavor of the questions addressed by Complexity Theory. This overview is quite vague, and is merely meant as a teaser. Most of the topics mentioned in it will be discussed at length in the various chapters of this book.

Complexity Theory is concerned with the study of the *intrinsic complexity* of computational tasks. Its "final" goals include the determination of the complexity of any well-defined task. Additional goals include obtaining an understanding of the relations between various computational phenomena (e.g., relating one fact regarding computational complexity to another). Indeed, we may say that the former type of goal is concerned with *absolute* answers regarding specific computational phenomena, whereas the latter type is concerned with questions regarding the *relation* between computational phenomena.

Interestingly, so far Complexity Theory has been more successful in coping with goals of the latter ("relative") type. In fact, the failure to resolve questions of the "absolute" type led to the flourishing of methods for coping with questions of the "relative" type. Musing for a moment, let us say that, in general, the difficulty of obtaining absolute answers may naturally lead to seeking conditional answers, which may in turn reveal interesting relations between phenomena. Furthermore, the lack of absolute understanding of individual phenomena seems to facilitate the development of methods for relating different phenomena. Anyhow, this is what happened in Complexity Theory.

Putting aside for a moment the frustration caused by the failure of obtaining absolute answers, we must admit that there is something fascinating in the success of relating different phenomena: In some sense, relations between phenomena are more revealing than absolute statements about individual phenomena. Indeed, the first example that comes to mind is the theory of NP-completeness. Let us consider this theory, for a moment, from the perspective of these two types of goals.

Complexity Theory has failed to determine the intrinsic complexity of tasks such as finding a satisfying assignment to a given (satisfiable) propositional formula or finding a 3-coloring of a given (3-colorable) graph. But it has succeeded in establishing that these two seemingly different computational tasks are in some sense the same (or, more precisely, are computationally equivalent). We find this success amazing and exciting, and hope that the reader shares these feelings. The same feeling of wonder and excitement is generated by many of the other discoveries of Complexity Theory. Indeed, the reader is invited to join a fast tour of some of the other questions and answers that make up the field of Complexity Theory.

We will indeed start with the *P versus NP Question*. Our daily experience is that it is harder to solve a problem than it is to check the correctness of a solution (e.g., think of either a puzzle or a research problem). Is this experience merely a coincidence or does it represent a fundamental fact of life (i.e., a property of the world)? Could you imagine a

[1] The quote is commonly interpreted as meaning that benefit arose out of misfortune.

world in which solving any problem is not significantly harder than checking a solution to it? Would the term "solving a problem" not lose its meaning in such a hypothetical (and impossible, in our opinion) world? The denial of the plausibility of such a hypothetical world (in which "solving" is not harder than "checking") is what "P different from NP" actually means, where P represents tasks that are efficiently solvable and NP represents tasks for which solutions can be efficiently checked.

The mathematically (or theoretically) inclined reader may also consider the task of proving theorems versus the task of verifying the validity of proofs. Indeed, finding proofs is a special type of the aforementioned task of "solving a problem" (and verifying the validity of proofs is a corresponding case of checking correctness). Again, "P different from NP" means that there are theorems that are harder to prove than to be convinced of their correctness when presented with a proof. This means that the notion of a "proof" is meaningful; that is, proofs do help when one is seeking to be convinced of the correctness of assertions. Here NP represents sets of assertions that can be efficiently verified with the help of adequate proofs, and P represents sets of assertions that can be efficiently verified from scratch (i.e., without proofs).

In light of the foregoing discussion it is clear that the P versus NP Question is a fundamental scientific question of far-reaching consequences. The fact that this question seems beyond our current reach led to the development of the theory of *NP-completeness*. Loosely speaking, this theory identifies a set of computational problems that are as hard as NP. That is, the fate of the P versus NP Question lies with each of these problems: If any of these problems is easy to solve then so are all problems in NP. Thus, showing that a problem is NP-complete provides evidence of its intractability (assuming, of course, "P different than NP"). Indeed, demonstrating the NP-completeness of computational tasks is a central tool in indicating hardness of natural computational problems, and it has been used extensively both in computer science and in other disciplines. We note that NP-completeness indicates not only the conjectured intractability of a problem but also its "richness" in the sense that the problem is rich enough to "encode" any other problem in NP. The use of the term "encoding" is justified by the exact meaning of NP-completeness, which in turn establishes relations between different computational problems (without referring to their "absolute" complexity).

The foregoing discussion of NP-completeness hints at *the importance of representation*, since it referred to different problems that encode one another. Indeed, the importance of representation is a central aspect of Complexity Theory. In general, Complexity Theory is concerned with problems for which the solutions are implicit in the problem's statement (or rather in the instance). That is, the problem (or rather its instance) contains all necessary information, and one merely needs to process this information in order to supply the answer.[2] Thus, Complexity Theory is concerned with manipulation of information, and its transformation from one representation (in which the information is given) to another representation (which is the one desired). Indeed, a solution to a computational problem is merely a different representation of the information given, that is, a representation in which the answer is explicit rather than implicit. For example, the answer to the question of whether or not a given Boolean formula is satisfiable is implicit in the formula itself (but the task is to make the answer explicit). Thus, Complexity Theory clarifies a central

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

issue regarding representation, that is, the distinction between what is explicit and what is implicit in a representation. Furthermore, it even suggests a quantification of the level of non-explicitness.

In general, Complexity Theory provides new viewpoints on various phenomena that were considered also by past thinkers. Examples include the aforementioned concepts of solutions, proofs, and representation as well as concepts like randomness, knowledge, interaction, secrecy, and learning. We next discuss the latter concepts and the perspective offered by Complexity Theory.

The concept of *randomness* has puzzled thinkers for ages. Their perspective can be described as ontological: They asked "what is randomness" and wondered whether it exists, at all (or is the world deterministic). The perspective of Complexity Theory is behavioristic: It is based on defining objects as equivalent if they cannot be told apart by any efficient procedure. That is, a coin toss is (defined to be) "random" (even if one believes that the universe is deterministic) if it is infeasible to predict the coin's outcome. Likewise, a string (or a distribution of strings) is "random" if it is infeasible to distinguish it from the uniform distribution (regardless of whether or not one can generate the latter). Interestingly, randomness (or rather pseudorandomness) defined this way is efficiently expandable; that is, under a reasonable complexity assumption (to be discussed next), short pseudorandom strings can be deterministically expanded into long pseudorandom strings. Indeed, it turns out that randomness is intimately related to intractability. Firstly, note that the very definition of pseudorandomness refers to intractability (i.e., the infeasibility of distinguishing a pseudorandomness object from a uniformly distributed object). Secondly, as stated, a complexity assumption, which refers to the existence of functions that are easy to evaluate but hard to invert (called *one-way functions*), implies the existence of deterministic programs (called *pseudorandom generators*) that stretch short random seeds into long pseudorandom sequences. In fact, it turns out that the existence of pseudorandom generators is equivalent to the existence of one-way functions.

Complexity Theory offers its own perspective on the concept of *knowledge* (and distinguishes it from information). Specifically, Complexity Theory views knowledge as the result of a hard computation. Thus, whatever can be efficiently done by anyone is not considered knowledge. In particular, the result of an easy computation applied to publicly available information is not considered knowledge. In contrast, the value of a hard-to-compute function applied to publicly available information is knowledge, and if somebody provides you with such a value then it has provided you with knowledge. This discussion is related to the notion of *zero-knowledge* interactions, which are interactions in which no knowledge is gained. Such interactions may still be useful, because they may convince a party of the *correctness* of specific data that was provided beforehand. For example, a zero-knowledge interactive proof may convince a party that a given graph is 3-colorable without yielding any 3-coloring.

The foregoing paragraph has explicitly referred to *interaction*, viewing it as a vehicle for gaining knowledge and/or gaining confidence. Let us highlight the latter application by noting that it may be easier to verify an assertion when allowed to interact with a prover rather than when reading a proof. Put differently, interaction with a good teacher may be more beneficial than reading any book. We comment that the added power of such *interactive proofs* is rooted in their being randomized (i.e., the verification procedure is randomized), because if the verifier's questions can be determined beforehand then the prover may just provide the transcript of the interaction as a traditional written proof.

Another concept related to knowledge is that of *secrecy*: Knowledge is something that one party may have while another party does not have (and cannot feasibly obtain by itself) – thus, in some sense knowledge is a secret. In general, Complexity Theory is related to *cryptography*, where the latter is broadly defined as the study of systems that are easy to use but hard to abuse. Typically, such systems involve secrets, randomness, and interaction as well as a complexity gap between the ease of proper usage and the infeasibility of causing the system to deviate from its prescribed behavior. Thus, much of cryptography is based on complexity theoretic assumptions and its results are typically transformations of relatively simple computational primitives (e.g., one-way functions) into more complex cryptographic applications (e.g., secure encryption schemes).

We have already mentioned the concept of *learning* when referring to learning from a teacher versus learning from a book. Recall that Complexity Theory provides evidence to the advantage of the former. This is in the context of gaining knowledge about publicly available information. In contrast, computational learning theory is concerned with learning objects that are only partially available to the learner (i.e., reconstructing a function based on its value at a few random locations or even at locations chosen by the learner). Complexity Theory sheds light on the intrinsic limitations of learning (in this sense).

Complexity Theory deals with a variety of computational tasks. We have already mentioned two fundamental types of tasks: *searching for solutions* (or rather "finding solutions") and *making decisions* (e.g., regarding the validity of assertions). We have also hinted that in some cases these two types of tasks can be related. Now we consider two additional types of tasks: *counting the number of solutions* and *generating random solutions*. Clearly, both the latter tasks are at least as hard as finding arbitrary solutions to the corresponding problem, but it turns out that for some natural problems they are not significantly harder. Specifically, under some natural conditions on the problem, approximately counting the number of solutions and generating an approximately random solution is not significantly harder than finding an arbitrary solution.

Having mentioned the notion of *approximation*, we note that the study of the complexity of finding "approximate solutions" is also of natural importance. One type of approximation problems refers to an objective function defined on the set of potential solutions: Rather than finding a solution that attains the optimal value, the approximation task consists of finding a solution that attains an "almost optimal" value, where the notion of "almost optimal" may be understood in different ways giving rise to different levels of approximation. Interestingly, in many cases, even a very relaxed level of approximation is as difficult to obtain as solving the original (exact) search problem (i.e., finding an approximate solution is as hard as finding an optimal solution). Surprisingly, these hardness-of-approximation results are related to the study of *probabilistically checkable proofs*, which are proofs that allow for ultra-fast probabilistic verification. Amazingly, every proof can be efficiently transformed into one that allows for probabilistic verification based on probing a *constant* number of bits (in the alleged proof). Turning back to approximation problems, we note that in other cases a reasonable level of approximation is easier to achieve than solving the original (exact) search problem.

Approximation is a natural relaxation of various computational problems. Another natural relaxation is the study of *average-case complexity*, where the "average" is taken over some "simple" distributions (representing a model of the problem's instances that may occur in practice). We stress that, although it was not stated explicitly, the entire discussion so far has referred to "worst-case" analysis of algorithms. We mention that worst-case complexity is a more robust notion than average-case complexity. For starters,

one avoids the controversial question of which instances are "important in practice" and correspondingly the selection of the class of distributions for which average-case analysis is to be conducted. Nevertheless, a relatively robust theory of average-case complexity has been suggested, albeit it is less developed than the theory of worst-case complexity.

In view of the central role of randomness in Complexity Theory (as evident, say, in the study of pseudorandomness, probabilistic proof systems, and cryptography), one may wonder as to whether the randomness needed for the various applications can be obtained in real life. One specific question, which received a lot of attention, is the possibility of "purifying" randomness (or "extracting good randomness from bad sources"). That is, can we use "defected" sources of randomness in order to implement almost perfect sources of randomness? The answer depends, of course, on the model of such defected sources. This study turned out to be related to Complexity Theory, where the most tight connection is between some type of *randomness extractors* and some type of pseudorandom generators.

So far we have focused on the time complexity of computational tasks, while relying on the natural association of efficiency with time. However, time is not the only resource one should care about. Another important resource is *space*: the amount of (temporary) memory consumed by the computation. The study of space complexity has uncovered several fascinating phenomena, which seem to indicate a fundamental difference between space complexity and time complexity. For example, in the context of space complexity, verifying proofs of validity of assertions (of any specific type) has the same complexity as verifying proofs of invalidity for the same type of assertions.

In case the reader feels dizzy, it is no wonder. We took an ultra-fast air tour of some mountain tops, and dizziness is to be expected. Needless to say, the rest of the book offers a totally different touring experience. We will climb some of these mountains by foot, step by step, and will often stop to look around and reflect.

Absolute Results (aka. Lower Bounds). As stated up-front, absolute results are not known for many of the "big questions" of Complexity Theory (most notably the P versus NP Question). However, several highly non-trivial absolute results have been proved. For example, it was shown that using negation can speed up the computation of monotone functions (which do not require negation for their mere computation). In addition, many promising techniques were introduced and employed with the aim of providing a low-level analysis of the progress of computation. However, as stated in the preface, the focus of this book is elsewhere.

1.1.2. Characteristics of Complexity Theory

We are successful because we use the right level of abstraction.
Avi Wigderson (1996)

Using the "right level of abstraction" seems to be a main characteristic of the theory of computation at large. The right level of abstraction means abstracting away second-order details, which tend to be context dependent, while using definitions that reflect the main issues (rather than abstracting them away, too). Indeed, using the right level of abstraction calls for an extensive exercising of good judgment, and one indication for having chosen the right abstractions is the result of their study.

One major choice, taken by the theory of computation at large, is the *choice of a model of computation and corresponding complexity measures and classes*. The choice, which is currently taken for granted, was to use a simple model that avoids both the extreme of being too realistic (and thus too detailed) as well as the extreme of being too abstract (and vague). On the one hand, the main model of computation (which is used in Complexity Theory) does not try to mimic (or mirror) the actual operation of real-life computers used at a specific historical time. Such a choice would have made it very hard to develop Complexity Theory as we know it and to uncover the fundamental relations discussed in this book: The mass of details would have obscured the view. On the other hand, avoiding any reference to any concrete model (like in the case of recursive function theory) does not encourage the introduction and study of natural measures of complexity. Indeed, as we shall see in Section 1.2.3, the choice was (and is) to use a simple model of computation (which does not mirror real-life computers), while avoiding any effects that are specific to that model (by keeping an eye on a host of variants and alternative models). The freedom from the specifics of the basic model is obtained by considering complexity classes that are invariant under a change of model (as long as the alternative model is "reasonable").

Another major choice is the use of *asymptotic analysis*. Specifically, we consider the complexity of an algorithm as a function of its input length, and study the asymptotic behavior of this function. It turns out that structure that is hidden by concrete quantities appears at the limit. Furthermore, depending on the case, we classify functions according to different criteria. For example, in the case of time complexity we consider classes of functions that are closed under multiplication, whereas in case of space complexity we consider closure under addition. In each case, the choice is governed by the nature of the complexity measure being considered. Indeed, one could have developed a theory without using these conventions, but this would have resulted in a far more cumbersome theory. For example, rather than saying that finding a satisfying assignment for a given formula is polynomial-time reducible to deciding the satisfiability of some other formulae, one could have stated the exact functional dependence of the complexity of the search problem on the complexity of the decision problem.

Both the aforementioned choices are common to other branches of the theory of computation. One aspect that makes Complexity Theory unique is its perspective on the most basic question of the theory of computation, that is, the way it studies the question of *what can be efficiently computed*. The perspective of Complexity Theory is general in nature. This is reflected in its primary focus on the relevant *notion of efficiency* (captured by corresponding resource bounds) rather than on specific computational problems. In most cases, complexity theoretic studies do not refer to any specific computational problems or refer to such problems merely as an illustration. Furthermore, even when specific computational problems are studied, this study is (explicitly or at least implicitly) aimed at understanding the computational limitations of certain resource bounds.

The aforementioned general perspective seems linked to the significant role of *conceptual considerations* in the field: The rigorous study of an intuitive notion of efficiency must be initiated with an adequate choice of definitions. Since this study refers to any possible (relevant) computation, the definitions cannot be derived by abstracting some concrete reality (e.g., a specific algorithmic schema). Indeed, the definitions attempt to capture any possible reality, which means that the choice of definitions is governed by conceptual principles and not merely by empirical observations.

1.1.3. Contents of This Book

This book is intended to serve as an introduction to Computational Complexity Theory. It consists of ten chapters and seven appendices, and can be used either as a textbook or for self-study. The chapters constitute the core of this book and are written in a style adequate for a textbook, whereas the appendices provide either relevant background or additional perspective and are written in the style of a survey article.

1.1.3.1. Overall Organization of the Book

Section 1.2 and Chapter 2 are a prerequisite for the rest of the book. Technically speaking, the notions and results that appear in these parts are extensively used in the rest of the book. More importantly, the former parts are the conceptual framework that shapes the field and provides a good perspective on the field's questions and answers. Indeed, Section 1.2 and Chapter 2 provide the very basic material that must be understood by anybody having an interest in Complexity Theory.

In contrast, the rest of the book covers more advanced material, which means that none of it can be claimed to be absolutely necessary for a basic understanding of Complexity Theory. In particular, although some advanced chapters refer to material in other advanced chapters, the relation between these chapters is not a fundamental one. Thus, one may choose to read and/or teach an arbitrary subset of the advanced chapters and do so in an arbitrary order, provided one is willing to follow the relevant references to some parts of other chapters (see Figure 1.1). Needless to say, we recommend reading and/or teaching all the advanced chapters, and doing so by following the order presented in this book.

As illustrated by Figure 1.1, some chapters (i.e., Chapters 3, 6, and 10) lump together topics that are usually presented separately. These decisions are related to our perspective on the corresponding topics.

Turning to the appendices, we note that some of them (e.g., Appendix G and parts of Appendices D and E) provide background information that is required in some of the advanced chapters. In contrast, other appendices (e.g., Appendices B and C and other parts of Appendices D and E) provide additional perspective that augments the advanced chapters. (The function of Appendices A and F will be clarified in §1.1.3.2.)

1.1.3.2. Contents of the Specific Parts

The rest of this section provides a brief summary of the contents of the various chapters and appendices. This summary is intended for the teacher and/or the expert, whereas the student is referred to the more novice-friendly summaries that appear in the book's preface.

Section 1.2: Preliminaries. This section provides the relevant background on computability theory, which is the basis for the rest of this book (as well as for Complexity Theory at large). Most importantly, it contains a discussion of central notions such as search and decision problems, algorithms that solve such problems, and their complexity. In addition, this section presents non-uniform models of computation (e.g., Boolean circuits).

Chapter 2: P, NP, and NP-completeness. This chapter presents the P-vs-NP Question both in terms of search problems and in terms of decision problems. The second main

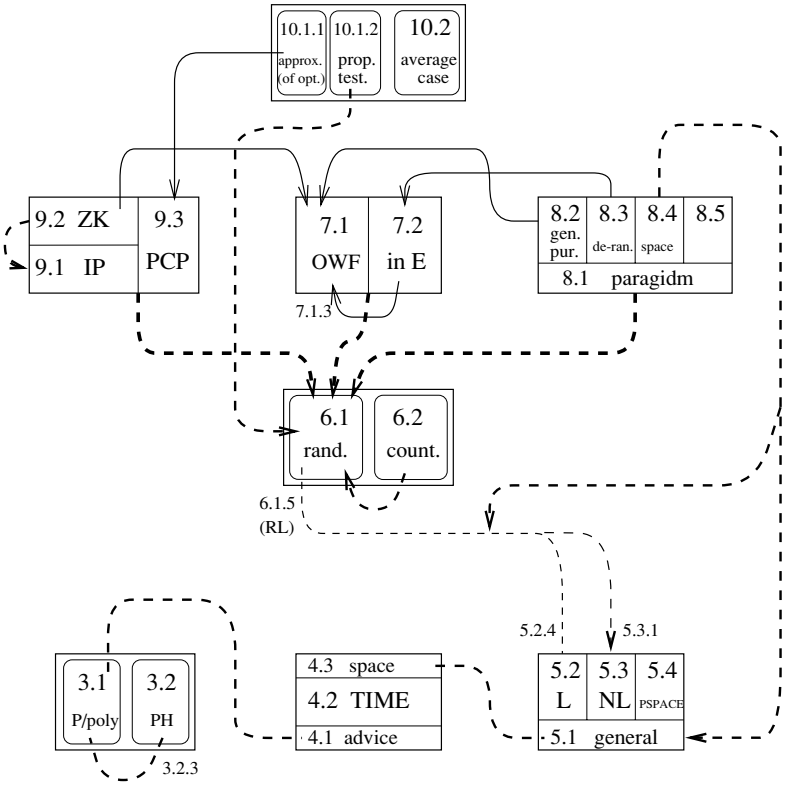


Figure 1.1: Dependencies among the advanced chapters. Solid arrows indicate the use of specific results that are stated in the section to which the arrow points. Dashed lines (and arrows) indicate an important conceptual connection; the wider the line, the tighter the connection. When relations are only between subsections, their index is indicated.

topic of this chapter is the theory of NP-completeness. The chapter also provides a treatment of the general notion of a (polynomial time) reduction, with special emphasis on self-reducibility. Additional topics include the existence of problems in NP that are neither NP-complete nor in P, optimal search algorithms, the class coNP, and promise problems.

Chapter 3: Variations on P and NP. This chapter provides a treatment of non-uniform polynomial time (P/poly) and of the Polynomial-time Hierarchy (PH). Each of the two classes is defined in two equivalent ways (e.g., P/poly is defined both in terms of circuits and in terms of "machines that take advice"). In addition, it is shown that if NP is contained in P/poly then PH collapses to its second level (i.e., Σ_2).

Chapter 4: More Resources, More Power? The focus of this chapter is on hierarchy theorems, which assert that typically more resources allow for solving more problems. These results depend on using bounding functions that can be computed without exceeding the amount of resources that they specify, and otherwise gap theorems may apply.

Chapter 5: Space Complexity. Among the results presented in this chapter are a log-space algorithm for testing connectivity of (undirected) graphs, a proof that $\mathcal{NL} = \text{co}\mathcal{NL}$,

and complete problems for $\mathcal{NL}$ and $\mathcal{PSPACE}$ (under log-space and poly-time reductions, respectively).

Chapter 6: Randomness and Counting. This chapter focuses on various randomized complexity classes (i.e., $\mathcal{BPP}$, $\mathcal{RP}$, and $\mathcal{ZPP}$) and the counting class $\#\mathcal{P}$. The results presented in this chapter include $\mathcal{BPP} \subset \mathcal{P}/\text{poly}$ and $\mathcal{BPP} \subseteq \Sigma_2$, the $\#\mathcal{P}$-completeness of the `Permanent`, the connection between approximate counting and uniform generation of solutions, and the randomized reductions of approximate counting to $\mathcal{NP}$ and of $\mathcal{NP}$ to solving problems with unique solutions.

Chapter 7: The Bright Side of Hardness. This chapter deals with two conjectures that are related to $\mathcal{P} \neq \mathcal{NP}$. The first conjecture is that there are problems in $\mathcal{E}$ that are not solvable by (non-uniform) families of small (say, polynomial-size) circuits, whereas the second conjecture is equivalent to the notion of *one-way functions*. Most of this chapter is devoted to "hardness amplification" results that convert these conjectures into tools that can be used for non-trivial derandomizations of $\mathcal{BPP}$ (resp., for a host of cryptographic applications).

Chapter 8: Pseudorandom Generators. The pivot of this chapter is the notion of *computational indistinguishability* and corresponding notions of pseudorandomness. The definition of general-purpose pseudorandom generators (running in polynomial time and withstanding any polynomial-time distinguisher) is presented as a special case of a general paradigm. The chapter also contains a presentation of other instantiations of the latter paradigm, including generators aimed at derandomizing complexity classes such as $\mathcal{BPP}$, generators withstanding space-bounded distinguishers, and some special-purpose generators.

Chapter 9: Probabilistic Proof Systems. This chapter provides a treatment of three types of probabilistic proof systems: *interactive proofs*, *zero-knowledge proofs*, and *probabilistic checkable proofs*. The results presented include $\mathcal{IP} = \mathcal{PSPACE}$, zero-knowledge proofs for any NP-set, and the PCP Theorem. For the latter, only overviews of the two different known proofs are provided.

Chapter 10: Relaxing the Requirements. This chapter provides a treatment of two types of approximation problems and a theory of average-case (or rather typical-case) complexity. The traditional type of approximation problem refers to search problems and consists of a relaxation of standard optimization problems. The second type is known as "property testing" and consists of a relaxation of standard decision problems. The theory of average-case complexity involves several non-trivial definitional choices (e.g., an adequate choice of the class of distributions).

Appendix A: Glossary of Complexity Classes. The glossary provides self-contained definitions of most complexity classes mentioned in the book.

Appendix B: On the Quest for Lower Bounds. The first part, devoted to Circuit Complexity, reviews lower bounds for the *size* of (restricted) circuits that solve natural computational problems. The second part, devoted to Proof Complexity, reviews lower bounds on the length of (restricted) propositional proofs of natural tautologies.