P, NP, and NP-Completeness

The Basics of Computational Complexity

The focus of this book is the P versus NP Question and the theory of NP-completeness. It also provides adequate preliminaries regarding computational problems and computational models.

The P versus NP Question asks whether finding solutions is harder than checking the correctness of solutions. An alternative formulation asks whether discovering proofs is harder than verifying their correctness. It is widely believed that the answer to these equivalent formulations is positive, and this is captured by saying that P is different from NP.

Although the P versus NP Question remains unresolved, the theory of NP-completeness offers evidence for the intractability of specific problems in NP by showing that they are universal for the entire class. Amazingly enough, NP-complete problems exist, and hundreds of natural computational problems arising in many different areas of mathematics and science are NP-complete.

ODED GOLDREICH is a Professor of Computer Science at the Weizmann Institute of Science and an Incumbent of the Meyer W. Weisgal Professorial Chair. He is an editor for the *SIAM Journal on Computing*, the *Journal of Cryptology*, and *Computational Complexity* and previously authored the books *Modern Cryptography, Probabilistic Proofs and Pseudorandomness*, the two-volume work *Foundations of Cryptography*, and *Computational Complexity: A Conceptual Perspective*.

P, NP, and NP-Completeness

The Basics of Computational Complexity

ODED GOLDREICH
Weizmann Institute of Science

CAMBRIDGE
UNIVERSITY PRESS

CAMBRIDGE UNIVERSITY PRESS
Cambridge, New York, Melbourne, Madrid, Cape Town, Singapore,
São Paulo, Delhi, Dubai, Tokyo, Mexico City

Cambridge University Press
32 Avenue of the Americas, New York, NY 10013-2473, USA

www.cambridge.org
Information on this title: www.cambridge.org/9780521122542

First published 2010

Printed in the United States of America

A catalog record for this publication is available from the British Library.

Library of Congress Cataloging in Publication data

Goldreich, Oded.
P, NP, and NP-completeness : the basics of computational complexity / Oded Goldreich.
p. cm.
Includes bibliographical references and index.
ISBN 978-0-521-19248-4 (hardback) – ISBN 978-0-521-12254-2 (pbk.)
1. Computational complexity. 2. Computer algorithms. 3. Approximation theory.
4. Polynomials. I. Title.
QA267.7.G652 2010
005.1–dc22 2010023587

ISBN 978-0-521-19248-4 Hardback
ISBN 978-0-521-12254-2 Paperback

to Dana

Contents

List of Figures *page* xi
Preface xiii
Overview xvii
To the Teacher xxi
Notations and Conventions xxv
Main Definitions and Results xxvii

1 Computational Tasks and Models **1**
Teaching Notes 2
1.1 Representation 3
1.2 Computational Tasks 5
1.2.1 Search Problems 5
1.2.2 Decision Problems 6
1.2.3 Promise Problems (an Advanced Comment) 8
1.3 Uniform Models (Algorithms) 8
1.3.1 Overview and General Principles 9
1.3.2 A Concrete Model: Turing Machines 11
1.3.2.1 The Actual Model 12
1.3.2.2 The Church-Turing Thesis 16
1.3.3 Uncomputable Functions 18
1.3.3.1 On the Existence of Uncomputable Functions 18
1.3.3.2 The Halting Problem 19
1.3.3.3 A Few More Undecidability Results 21
1.3.4 Universal Algorithms 22
1.3.4.1 The Existence of Universal Algorithms 23
1.3.4.2 A Detour: Kolmogorov Complexity 24
1.3.5 Time (and Space) Complexity 26
1.3.6 Oracle Machines and Turing-Reductions 29

1.3.7 Restricted Models 31
1.4 Non-Uniform Models (Circuits and Advice) 31
1.4.1 Boolean Circuits 32
1.4.1.1 The Basic Model 32
1.4.1.2 Circuit Complexity 35
1.4.2 Machines That Take Advice 36
1.4.3 Restricted Models 37
1.4.3.1 Boolean Formulae 38
1.4.3.2 Other Restricted Classes of Circuits 39
1.5 Complexity Classes 40
Exercises 41

2 The P versus NP Question 48
Teaching Notes 49
2.1 Efficient Computation 50
2.2 The Search Version: Finding versus Checking 53
2.2.1 The Class P as a Natural Class of Search Problems 54
2.2.2 The Class NP as Another Natural Class of Search Problems 56
2.2.3 The P versus NP Question in Terms of Search Problems 57
2.3 The Decision Version: Proving versus Verifying 58
2.3.1 The Class P as a Natural Class of Decision Problems 59
2.3.2 The Class NP and NP-Proof Systems 59
2.3.3 The P versus NP Question in Terms of Decision Problems 62
2.4 Equivalence of the Two Formulations 63
2.5 Technical Comments Regarding NP 65
2.6 The Traditional Definition of NP 66
2.7 In Support of P Being Different from NP 69
2.8 Philosophical Meditations 70
Exercises 71

3 Polynomial-time Reductions 74
Teaching Notes 75
3.1 The General Notion of a Reduction 75
3.1.1 The Actual Formulation 76
3.1.2 Special Cases 77
3.1.3 Terminology and a Brief Discussion 79
3.2 Reducing Optimization Problems to Search Problems 81
3.3 Self-Reducibility of Search Problems 83

3.3.1 Examples 85
3.3.2 Self-Reducibility of NP-Complete Problems 87
3.4 Digest and General Perspective 88
Exercises 89

4 NP-Completeness 96
Teaching Notes 97
4.1 Definitions 98
4.2 The Existence of NP-Complete Problems 99
Bounded Halting and Non-Halting 102
4.3 Some Natural NP-Complete Problems 103
4.3.1 Circuit and Formula Satisfiability: CSAT and SAT 104
4.3.1.1 The NP-Completeness of CSAT 105
4.3.1.2 The NP-Completeness of SAT 109
4.3.2 Combinatorics and Graph Theory 113
4.3.3 Additional Properties of the Standard Reductions 120
4.3.4 On the Negative Application of NP-Completeness 121
4.3.5 Positive Applications of NP-Completeness 122
4.4 NP Sets That Are Neither in P nor NP-Complete 126
4.5 Reflections on Complete Problems 130
Exercises 133

5 Three Relatively Advanced Topics 142
Teaching Notes 142
5.1 Promise Problems 142
5.1.1 Definitions 143
5.1.1.1 Search Problems with a Promise 143
5.1.1.2 Decision Problems with a Promise 144
5.1.1.3 Reducibility Among Promise Problems 145
5.1.2 Applications and Limitations 146
5.1.2.1 Formulating Natural Computational Problems 146
5.1.2.2 Restricting a Computational Problem 147
5.1.2.3 Non-generic Applications 147
5.1.2.4 Limitations 148
5.1.3 The Standard Convention of Avoiding Promise Problems 149
5.2 Optimal Search Algorithms for NP 151
5.3 The Class coNP and Its Intersection with NP 154
Exercises 158

Historical Notes 165

Epilogue: A Brief Overview of Complexity Theory **169**

Appendix: Some Computational Problems **177**
A.1 Graphs 177
A.2 Boolean Formulae 179

Bibliography 181
Index 183

List of Figures

0.1	Outline of the suggested course.	*page* xxiv
1.1	A single step by a Turing machine.	12
1.2	Multiple steps of the machine depicted in Figure 1.1.	15
1.3	A circuit computing $f(x_1, x_2, x_3, x_4) = (x_1 \oplus x_2, x_1 \wedge \neg x_2 \wedge x_4)$.	34
1.4	Recursive construction of parity circuits and formulae.	38
1.5	A 3DNF formula computing $x_1 \oplus x_2 \oplus x_3$.	39
2.1	Solving S by using a solver for R.	64
2.2	Solving R by using a solver for S'_R.	65
3.1	The Cook-reduction that arises from a Karp-reduction.	78
3.2	The Cook-reduction that arises from a Levin-reduction.	80
3.3	The three proofs of Theorem 3.8.	95
4.1	Overview of the emulation of a computation by a circuit.	106
4.2	Consecutive computation steps of a Turing machine.	107
4.3	The idea underlying the reduction of CSAT to SAT.	111
4.4	The reduction to G3C – the clause gadget and its sub-gadget.	119
4.5	The reduction to G3C – connecting the gadgets.	120
4.6	The (non-generic) reductions presented in Section 4.3.	121
5.1	A schematic depiction of a promise problem.	145
5.2	The world view under $\mathcal{P} \neq \text{co}\mathcal{NP} \cap \mathcal{NP} \neq \mathcal{NP}$.	158

Preface

The quest for efficiency is ancient and universal, as time and other resources are always in shortage. Thus, the question of which tasks can be performed efficiently is central to the human experience.

A key step toward the systematic study of the aforementioned question is a rigorous definition of the notion of a task and of procedures for solving tasks. These definitions were provided by computability theory, which emerged in the 1930s. This theory focuses on computational tasks, considers automated procedures (i.e., computing devices and algorithms) that may solve such tasks, and studies the class of solvable tasks.

In focusing attention on computational tasks and algorithms, computability theory has set the stage for the study of the computational resources (like time) that are required by such algorithms. When this study focuses on the resources that are necessary for *any* algorithm that solves a particular task (or a task of a particular type), it is viewed as belonging to the theory of Computational Complexity (also known as Complexity Theory). In contrast, when the focus is on the design and analysis of specific algorithms (rather than on the intrinsic complexity of the task), the study is viewed as belonging to a related area that may be called Algorithmic Design and Analysis. Furthermore, Algorithmic Design and Analysis tends to be sub-divided according to the domain of mathematics, science, and engineering in which the computational tasks arise. In contrast, Complexity Theory typically maintains a unity of the study of computational tasks that are solvable within certain resources (regardless of the origins of these tasks).

Complexity Theory is a central field of the theoretical foundations of computer science (CS). It is concerned with the study of the *intrinsic complexity of computational tasks*. That is, a typical Complexity theoretic study refers to the computational resources required to solve a computational task (or a class of such tasks), rather than referring to a specific algorithm or an algorithmic

schema. Actually, research in Complexity Theory tends to *start with and focus on the computational resources themselves*, and addresses the effect of limiting these resources on the class of tasks that can be solved. Thus, Computational Complexity is the general study of what can be achieved within limited time (and/or other limitations on natural computational resources).

The most famous question of Complexity Theory is the P-vs-NP Question. This question can be phrased as asking whether *finding* solutions to certain problems is harder than *checking* the correctness of solutions to these problems. Indeed, this phrasing refers to so-called search problems (i.e., problems of searching for solutions). An alternative phrasing, which refers to so-called decision problems, asks whether or not deciding the validity of assertions can be facilitated by the presentation of adequate proofs. Equivalently, the question is whether discovering proofs (of the validity of assertions) is harder than verifying their correctness; that is, is proving harder than verifying?

The fundamental nature of the P-vs-NP Question is evident in each of the foregoing formulations, which are in fact equivalent. It is widely believed that the answer to these equivalent formulations is that finding (resp., proving) is harder than checking (resp., verifying); that is, *it is believed that P is different from NP*, where P corresponds to the class of efficiently solvable problems and NP corresponds to the seemingly wider class of problems allowing for efficient verification of potential solutions.

Indeed, the P-vs-NP Question has been unresolved since the early 1970s, and it is the author's guess that the question will remain unresolved for centuries, waiting for the development of a deeper understanding of the nature of efficient computation. However, life will continue in the meantime, and it will bring along a variety of NP-problems, where some of these problems will be placed in P (by presenting efficient algorithms solving them) and others will resist such attempts and will be conjectured to be too computationally hard to belong to P. Actually, the latter description is not a wild guess; this has been the state of affairs for several decades now.

At present, when faced with a seemingly hard problem in NP, we can only hope to prove that it is not in P by assuming that NP is different from P. Thus, we seek ways of proving that if the problem at hand is in P, then NP equals P, which means that all problems in NP are in P. This is where the theory of NP-completeness comes into the picture. Intuitively, a problem in NP is called NP-complete if any efficient algorithm for it can be converted into an efficient algorithm for any other problem in NP. It follows that if some NP-complete problem is in P, then all problems in NP are in P. Hence, if NP is different from P, then no NP-complete problem can be in P. Consequently, the P-vs-NP

Question is captured by the question of whether or not an individual (NP-complete) problem can be solved efficiently. Amazingly enough, NP-complete problems exist, and furthermore, hundreds of natural computational problems arising in many different areas of mathematics and science are NP-complete.

The aforementioned conversion of an efficient algorithm for one problem into efficient algorithms for other problems is actually performed by a translation of the latter problems' instances. Such a translation is called a reduction, and the theory of NP-completeness is based on the notion of efficient reductions. In general, one computational problem is (efficiently) reducible to another problem if it is possible to (efficiently) solve the former when provided access to an (efficient) algorithm for solving the latter. A problem (in NP) is NP-complete if any problem in NP is efficiently reducible to it, which means that *each individual NP-complete problem "encodes" all problems in NP*. The fact that NP-complete problems exist, let alone in such an abundance and variety, is indeed amazing.

Since its discovery, NP-completeness has been used as the main tool by which the intrinsic complexity of certain problems is demonstrated. A vast number of NP-completeness results have been discovered since the early 1970s. These discoveries have been guiding theoretical research as well as technological development by indicating when one needs to relax computational problems in order to obtain efficient procedures. This impact is neither confined to computer science nor to the need to solve some computational problems. It typically occurs when researchers or engineers seek a simple characterization of objects that satisfy some property, whereas it turns out that deciding whether a given object has this property is an NP-complete problem. Needless to say, in such a case, no *simple* characterization is likely to exist, and so one better abandon the search for it. Indeed, diverse scientific disciplines, which were unsuccessfully struggling with some of their internal questions, came to realize that these questions are inherently difficult since they are closely related to computational problems that are NP-complete.

The Current Book. The main focus of the current book is on the P-vs-NP Question and on the theory of NP-completeness. Indeed, a large portion of the book is devoted to presenting and studying the various formulations of the P-vs-NP Question. This portion may be viewed as a mathematical articulation of the intuitive gap between searching for solutions and checking their validity (or between proving theorems and verifying the correctness of proofs). Another large portion of the book is devoted to the presentation of the theory of NP-completeness, while providing a treatment of the general notion of efficient

reductions between computational problems. This portion may be viewed as a mathematical articulation of the daily notion of a "reduction" (i.e., solving one problem by using a known procedure for another problem), augmented with the fundamental and surprising feature of "universality" (i.e., the existence of complete problems to which all problems can be reduced).

The book, which includes adequate preliminaries regarding computational problems and computational models, aims to provide a wide perspective on the issues in its core. For example, the treatment of efficient reductions goes beyond the minimum that suffices for a presentation of the theory of NP-completeness, and this feature supports the study of the relative complexity of search and decision problems. In general, the book is believed to present the very basics of Complexity Theory, while bearing in mind that most readers do not intend to specialize in Complexity Theory (and yet hoping that some will be motivated to do so).

Relation to a Different Book by the Author. The current book is a significant revision of Chapter 2 (and Section 1.2) of the author's book *Computational Complexity: A Conceptual Perspective* [13]. The revision was aimed at making the book more friendly to the novice. In particular, numerous technical expositions were further detailed and many exercises were added.

Web Site for Notices Regarding This Book. The author intends to maintain a Web site listing corrections of various types. The location of the site is

`http://www.wisdom.weizmann.ac.il/~oded/bc-book.html`

Acknowledgments. The author is grateful to Asilata Bapat and Michael Forbes for their careful reading of a draft of this book and for the numerous corrections and suggestions that they provided.

Overview

This book starts by providing the relevant background on *computability theory*, which is the setting in which Complexity theoretic questions are being studied. Most importantly, this preliminary chapter (i.e., Chapter 1) provides a treatment of central notions, such as search and decision problems, algorithms that solve such problems, and their complexity. Special attention is given to the notion of a universal algorithm.

The main part of this book (i.e., Chapters 2–5) focuses on the P-vs-NP Question and on the theory of NP-completeness. Additional topics covered in this part include the general notion of an efficient reduction (with a special emphasis on reductions of search problems to corresponding decision problems), the existence of problems in NP that are neither NP-complete nor in P, the class coNP, optimal search algorithms, and promise problems. A brief overview of this main part follows.

The P-vs-NP Question. Loosely speaking, the P-vs-NP Question refers to search problems for which the correctness of solutions can be efficiently checked (i.e., there is an efficient algorithm that given a solution to a given instance determines whether or not the solution is correct). Such search problems correspond to the class NP, and the P-vs-NP Question corresponds to whether or not all these search problems can be solved efficiently (i.e., is there an efficient algorithm that given an instance finds a correct solution). Thus, the P-vs-NP Question can be phrased as asking *whether finding solutions is harder than checking the correctness of solutions*.

An alternative formulation, in terms of decision problems, refers to assertions that have efficiently verifiable proofs (of relatively short length). Such sets of assertions also correspond to the class NP, and the P-vs-NP Question corresponds to whether or not proofs for such assertions can be found efficiently (i.e., is there an efficient algorithm that given an assertion determines

its validity and/or finds a proof for its validity?). Thus, the P-vs-NP Question can also be phrased as asking *whether discovering proofs is harder than verifying their correctness*; that is, is proving harder than verifying (or are proofs valuable at all).

In these equivalent formulations of the P-vs-NP Question, P corresponds to the class of efficiently solvable problems, whereas NP corresponds to a natural class of problems for which it is reasonable to seek efficient solvability (i.e., NP corresponds to the seemingly wider class of problems allowing for efficient verification of potential solutions). We also note that in both cases, equality between P and NP contradicts our intuitions regarding the notions that underlie the formulation of NP (i.e., the notions of solving search problems and proving theorems).

Indeed, it is widely believed that the answer to these two equivalent formulations of the P-vs-NP Question is that *P is different from NP*; that is, finding (resp., discovering) is harder than checking (resp., verifying). The fact that this natural conjecture is unsettled seems to be one of the big sources of frustration of Complexity Theory. The author's opinion, however, is that this feeling of frustration is unjustified and is rooted in unrealistic expectations (i.e., naive underestimations of the difficulty of relating complexity classes of such a nature). In any case, at present, when faced with a seemingly hard problem in NP, we cannot expect to prove that the problem is not in P unconditionally. The best we can expect is a conditional proof that the said problem is not in P, based on the assumption that NP is different from P. The contrapositive is proving that if the said problem is in P, then so is any problem in NP (i.e., NP equals P). The theory of NP-completeness captures this idea.

NP-Completeness. The theory of NP-completeness is based on the notion of an efficient reduction, which is a relation between computational problems. Loosely speaking, one computational problem is efficiently reducible to another problem if it is possible to efficiently solve the former when provided with an (efficient) algorithm for solving the latter. Thus, the first problem is not harder to solve than the second one. A problem (in NP) is NP-complete if any problem in NP is efficiently reducible to it, which means that the first problem "encodes" all problems in NP (and so, in some sense, is the hardest among them). Indeed, the fate of the entire class NP (with respect to inclusion in P) rests with each individual NP-complete problem. In particular, showing that a problem is NP-complete implies that this problem is not in P unless NP equals P.

The fact that NP-complete problems can be defined does not mean that they exist. Indeed, the ability of an individual problem to encode all problems in a class as diverse as NP is unfamiliar in daily life, and a layperson is likely to guess

that such a phenomenon is self-contradictory (especially when being told that the complete problem has to be in the same class). Nevertheless, NP-complete problems exist, and furthermore, hundreds of natural computational problems arising in many different areas of mathematics and science are NP-complete.

The list of known NP-complete problems includes finding a satisfiable assignment to a given Boolean formula (or deciding whether such an assignment exists), finding a 3-coloring of the vertices of a given graph (or deciding whether such a coloring exists), and so on. The core of establishing the NP-completeness of these problems is showing that each of them can encode any other problem in NP. Thus, these demonstrations provide a method of encoding instances of any NP problem as instances of the target NP-complete problem.

The Actual Organization. The foregoing paragraphs refer to material that is covered in Chapters 2–4. Specifically, Chapter 2 is devoted to the P-vs-NP Question per se, Chapter 3 is devoted to the notion of an efficient reduction, and Chapter 4 is devoted to the theory of NP-completeness. We mention that NP-complete problems *are not the only seemingly hard problems in NP*; that is, if P is different from NP, then NP contains problems that are neither NP-complete nor in P (see Section 4.4).

Additional related topics are discussed in Chapter 5. In particular, in Section 5.2, it is shown that the P-vs-NP Question is not about inventing sophisticated algorithms or ruling out their existence, but rather boils down to the analysis of a single known algorithm; that is, we will present an optimal search algorithm for any problem in NP, while having no clue about its time-complexity.

Each of the main chapters (i.e., Chapters 1–4) starts with a short overview, which sets the stage for the entire chapter. These overviews provide the basic motivation for the notions defined, as well as a high-level summary of the main results, and hence should not be skipped. The chapter's overview is followed by teaching notes, which assume familiarity with the material and thus are better skipped by the novice. Each chapter ends with exercises, which are designed to help verify the basic understanding of the main text (and not to test or inspire creativity). In a few cases, exercises (augmented by adequate guidelines) are used for presenting related advanced material.

The book also includes a short historical account (see Historical Notes), a brief overview of Complexity Theory at large (see Epilogue), and a laconic review of some popular computational problems (see Appendix).

To the Teacher

According to a common opinion, the most important aspect of a scientific work is the technical result that it achieves, whereas explanations and motivations are merely redundancy introduced for the sake of "error correction" and/or comfort. It is further believed that, as with a work of art, the interpretation of the work should be left to the reader.

The author strongly disagrees with the aforementioned opinions, and argues that there is a fundamental difference between art and science, and that this difference refers exactly to the meaning of a piece of work. Science is concerned with meaning (and not with form), and in its quest for truth and/or understanding, science follows philosophy (and not art). The author holds the opinion that the most important aspects of a scientific work are the intuitive question that it addresses, the reason that it addresses this question, the way it phrases the question, the approach that underlies its answer, and the ideas that are embedded in the answer. Following this view, it is important to communicate these aspects of the work.

The foregoing issues are even more acute when it comes to Complexity Theory, firstly because conceptual considerations seem to play an even more central role in Complexity Theory than in other scientific fields. Secondly (and even more importantly), Complexity Theory is extremely rich in conceptual content. Thus, communicating this content is of primary importance, and failing to do so misses the most important aspects of Complexity Theory.

Unfortunately, the conceptual content of Complexity Theory is rarely communicated (explicitly) in books and/or surveys of the area. The annoying (and quite amazing) consequences are students who have only a vague understanding of the *meaning* and general relevance of the fundamental notions and results that they were taught. The author's view is that these consequences are easy to avoid by taking the time to explicitly discuss the *meaning* of definitions and results. A closely related issue is using the "right" definitions (i.e., those that

reflect better the fundamental nature of the notion being defined) and emphasizing the (conceptually) "right" results. The current book is written accordingly; two concrete and central examples follow.

The first example refers to the presentation of the P-vs-NP Question, where we avoid using (polynomial-time) non-deterministic machines. We believe that these fictitious "machines" have a negative effect from both a conceptual and a technical point of view. The conceptual damage caused by defining NP in terms of (polynomial-time) non-deterministic machines is that it is unclear why one should care about what such machines can do. Needless to say, the reason to care is clear when noting that these fictitious "machines" offer a (convenient but rather slothful) way of phrasing fundamental issues. The technical damage caused by using non-deterministic machines is that they tend to confuse the students.

In contrast to using a fictitious model as a pivot, we define NP in terms of proof systems such that the fundamental nature of this class and the P-vs-NP Question are apparent. We also push to the front a formulation of the P-vs-NP Question in terms of search problems. We believe that this formulation may appeal to non-experts even more than the formulation of the P-vs-NP Question in terms of decision problems. The aforementioned formulation refers to classes of search problems that are analogous to the decision problem classes P and NP. Specifically, we consider the classes $\mathcal{PF}$ and $\mathcal{PC}$ (see Definitions 2.2 and 2.3), where $\mathcal{PF}$ consists of search problems that are efficiently solvable and $\mathcal{PC}$ consists of search problems having efficiently checkable solutions.[1]

To summarize, we suggest presenting the P-vs-NP Question both in terms of search problems and in terms of decision problems. Furthermore, when presenting the decision-problem version, we suggest introducing NP by explicitly referring to the terminology of proof systems (rather than using the more standard formulation, which is based on non-deterministic machines). We mention that the formulation of NP as proof systems is also a better starting point for the study of more advanced issues (e.g., counting classes, let alone probabilistic proof systems).

Turning to the second example, which refers to the theory of NP-completeness, we highlight a central recommendation regarding the presentation of this theory. We believe that from a conceptual point of view, the mere existence of NP-complete problems is an amazing fact. We thus suggest emphasizing and discussing this fact per se. In particular, we recommend first proving the mere existence of NP-complete problems, and only later establishing the fact that certain natural problems such as SAT are NP-complete. Also, when establishing the NP-completeness of SAT, we recommend decoupling

[1] Indeed, these classes are often denoted $\mathcal{FP}$ and $\mathcal{FNP}$, respectively.

the emulation of Turing machines by circuits (used for establishing the NP-completeness of CSAT) from the emulation of circuits by formulae (used in the reduction of CSAT to SAT).

Organization. In Chapter 1, we present the basic framework of Computational Complexity, which serves as a stage for the rest of the book. In particular, we formalize the notions of search and decision problems (see Section 1.2), algorithms solving them (see Section 1.3), and their time complexity (see Section 1.3.5). In Chapter 2, we present the two formulations of the P-vs-NP Question. The general notion of a reduction is presented in Chapter 3, where we highlight its applicability outside the domain of NP-completeness. In particular, in Section 3.3 we treat reductions of search problems to corresponding decision problems. Chapter 4 is devoted to the theory of NP-completeness, whereas Chapter 5 treats three relatively advanced topics (i.e., the framework of promise problems, the existence of optimal search algorithms for NP, and the class coNP). The book ends with an Epilogue, which provides a brief overview of Complexity Theory, and an Appendix that reviews some popular computational problems (which are used as examples in the main text).

The Chapters' Overviews. Each of the main chapters (i.e., Chapters 1–4) starts with a short overview, which provides the basic motivation for the notions defined in that chapter as well as a high-level summary of the chapter's main results. We suggest using these overviews as a basis for motivational discussions preceding the actual technical presentation.

Additional Teaching Notes. Each chapter overview is followed by additional teaching notes. These notes articulate various choices made in the presentation of the material in the corresponding chapter.

Basing a Course on the Current Book. The book can serve as a basis for an undergraduate course, which may be called *Basics of Computational Complexity*. The core material for this course is provided by Chapters 1–4. Specifically, Sections 1.1–1.3 provide the required elements of computability theory, and Chapters 2–4 provide the basic elements of Complexity Theory. In addition, §1.4.1.1 and §1.4.3.1 (or, alternatively, Appendix A.2) provide preliminaries regarding Boolean circuits and formulae that are required in Section 4.3 (which refers to CSAT and SAT). For a schematic outline of the course, see Figure 0.1.

On the Choice of Additional (Basic and Advanced) Topics. As depicted in Figure 0.1, depending on time constraints, we suggest augmenting the core material with a selection of additional basic and advanced topics. As for

TOPIC	SECTIONS
Elements of computability theory	1.1–1.3
The P-vs-NP Question	2.1–2.4, 2.7
Optional: definitional variations	2.5, 2.6
Polynomial-time reductions	3.1–3.3
The existence of NP-complete problems	4.1–4.2
Natural NP-complete problems (e.g., CSAT, SAT, VC)	4.3
Preliminaries on Boolean circuits and formulae	1.4.1, 1.4.3, A.2
Add'l basic topics: NPI, promise problems, optimal search	4.4, 5.1, 5.2
Advanced topics, if time permits	from [13, 1]

Figure 0.1. Outline of the suggested course.

the basic topics, we recommend at least mentioning the class NPI, promise problems, and the optimal search algorithms for NP. Regarding the choice of advanced topics, we recommend an introduction to probabilistic proof systems. In our opinion, this choice is most appropriate because it provides natural extensions of the notion of an NP-proof system and offers very appealing positive applications of NP-completeness. Section 4.3.5 provides a brief overview of probabilistic proof systems, while [13, Chap. 9] provides an extensive overview (which transcends the needs of a basic complexity course). Alternative advanced topics can be found in [13, 1].

A Revision of the CS Curriculum. The best integration of the aforementioned course in undergraduate CS education calls for a revision of the standard CS curriculum. Indeed, we believe that there is no real need for a semester-long course in *Computability* (i.e., a course that focuses on what can be computed rather than on what can be computed efficiently). Instead, CS undergraduates should take a course in Computational Complexity, which should contain the computability aspects that serve as a basis for the study of efficient computation (i.e., the rest of this course). Specifically, the computability aspects should occupy at most one-third of the course, and the focus should be on basic complexity issues (captured by P, NP, and NP-completeness), which may be augmented by a selection of some more advanced material. Indeed, such a course can be based on the current book (possibly augmented by a selection of some additional topics from, say, [13, 1]).

Notations and Conventions

Although we do try to avoid using various notations and conventions that may not be familiar to the reader, some exceptions exists – especially in advanced discussions. In order to be on the safe side, we list here some standard notations and conventions that are (lightly) used in the book.

Standard Asymptotic Notation. When referring to integral functions, we use the standard asymptotic notation; that is, for $f, g : \mathbb{N} \to \mathbb{N}$, we write $f = O(g)$ if there exists a constant $c > 0$ such that $f(n) \leq c \cdot g(n)$ holds for all sufficiently large $n \in \mathbb{N}$. We usually denote by "poly" an unspecified polynomial, and write $f(n) = \text{poly}(n)$ instead of "there exists a polynomial p such that $f(n) \leq p(n)$ for all $n \in \mathbb{N}$."

Standard Combinatorial and Graph Theory Terms and Notation. For a natural number $n \in \mathbb{N}$, we denote $[n] \stackrel{\text{def}}{=} \{1, \ldots, n\}$. Many of the computational problems that we mention refer to finite (undirected) graphs. Such a graph, denoted $G = (V, E)$, consists of a set of vertices, denoted V, and a set of edges, denoted E, which are unordered pairs of vertices. By default, graphs are undirected, whereas directed graphs consist of vertices and directed edges, where a directed edge is an order pair of vertices. For further background on graphs and computational problems regarding graphs, the reader is referred to Appendix A.1.

Typographic Conventions. We denote formally defined complexity classes by calligraphic letters (e.g., $\mathcal{NP}$), but we do so only after defining these classes. Furthermore, when we wish to maintain some ambiguity regarding the specific formulation of a class of problems, we use Roman font (e.g., NP may denote either a class of search problems or a class of decision problems). Likewise,

we denote formally defined computational problems by typewriter font (e.g., SAT). In contrast, generic problems and algorithms will be denoted by italic font.

Our Use of Footnotes. In trying to accommodate a diverse spectrum of readers, we use footnotes for presentation of additional details that most readers may wish to skip but some readers may find useful. The most common usage of footnotes is for providing additional technical details that may seem obvious to most readers but be missed by some others. Occasionally, footnotes are also used for advanced comments.

Main Definitions and Results

Following is a list of the main definitions and results presented in the book. The list only provides a laconic description of each of the items, while a full description can be found in the actual text (under the provided reference). The list is ordered approximately according to the order of appearance of the corresponding topics in the main text.

Search and Decision Problems. The former refer to finding solutions to given instances, whereas the latter refer to determining whether the given instance has a predetermined property. See Definitions 1.1 and 1.2, respectively.

Turing Machines. The model of Turing machines offers a relatively simple formulation of the notion of an algorithm. See Section 1.3.2.

Theorem 1.4. *The set of computable functions is countable, whereas the set of all functions (from strings to strings) is not countable.*

Theorem 1.5. *The Halting Problem is undecidable.*

Universal Algorithms. A universal machine computes the partial function $\mathtt{u}$ that is defined on pairs $(\langle M\rangle, x)$ such that M halts on input x, in which case it holds that $\mathtt{u}(\langle M\rangle, x) = M(x)$. See Section 1.3.4.

Efficient and Inefficient. Efficiency is associated with polynomial-time computations, whereas computations requiring more time are considered inefficient or intractable (or infeasible). See Section 2.1.

The Class $\mathcal{PF}$ (Polynomial-time Find). The class of efficiently solvable search problems. See Definition 2.2.

The Class $\mathcal{PC}$ (Polynomial-time Check). The class of search problems having efficiently checkable solutions. See Definition 2.3.

The Notations S_R and $R(x)$ Associated with a Search Problem R. For any search problem, R, we denote the set of solutions to the instance x by $R(x)$ (i.e., $R(x) = \{y : (x, y) \in R\}$), and denote the set of instances having solutions by S_R (i.e., $S_R = \{x : R(x) \neq \emptyset\}$).

The Class $\mathcal{P}$. The class of efficiently solvable decision problems. See Definition 2.4.

The Class $\mathcal{NP}$. The class of decision problems having efficiently verifiable proof systems. See Definition 2.5.

Theorem 2.6. *$\mathcal{PC} \subseteq \mathcal{PF}$ if and only if $\mathcal{P} = \mathcal{NP}$.*

The P-vs-NP Question. It is widely believed that P is different from NP. This belief is supported by both philosophical and empirical considerations. See Section 2.7.

The Traditional Definition of $\mathcal{NP}$. Traditionally, $\mathcal{NP}$ is defined as the class of sets that can be decided by a *fictitious* device called a non-deterministic polynomial-time machine (which explains the source of the notation NP). See Section 2.6.

Cook-reductions. A problem Π is Cook-reducible to a problem Π' if Π can be solved efficiently when given access to any procedure (or oracle) that solves the problem Π'. See Definition 3.1.

Karp-reductions. A decision problem S is Karp-reducible to a decision problem S' if there exists a polynomial-time computable function f such that, for every x, it holds that $x \in S$ if and only if $f(x) \in S'$. See Definition 3.3.

Levin-reductions. A search problem R is Levin-reducible to a search problem R' if there exists polynomial-time computable functions f and g such that (1) f is a Karp-reduction of S_R to $S_{R'}$, and (2) for every $x \in S_R$ and $y' \in R'(f(x))$ it holds that $(x, g(x, y')) \in R$. See Definition 3.4.

Theorem 3.2. *Every search problem in $\mathcal{PC}$ is Cook-reducible to some decision problem in $\mathcal{NP}$.*

Self-reducibility of Search Problems. The decision implicit in a search problem R is deciding membership in the set S_R, and R is called self-reducible if it is Cook-reducible to S_R. See Section 3.3.

NP-Completeness (of Decision Problems). A decision problem S is $\mathcal{NP}$-complete if (1) S is in $\mathcal{NP}$, and (2) every decision problem in $\mathcal{NP}$ is Karp-reducible to S. See Definition 4.1.

NP-Completeness of Search Problems. A search problem R is $\mathcal{PC}$-complete (or NP-complete) if (1) R is in $\mathcal{PC}$, and (2) every search problem in $\mathcal{PC}$ is Levin-reducible to R. See Definition 4.2.

Theorem 4.3. *There exist NP-complete search and decision problems.*

Theorems 4.5 and 4.6 (Also Known as Cook–Levin Theorem). Circuit satisfiability (CSAT) and formula satisfiability (SAT) are NP-complete.

Proposition 4.4. *If an $\mathcal{NP}$-complete decision problem S is Karp-reducible to a decision problem $S' \in \mathcal{NP}$* (resp., a $\mathcal{PC}$-complete search problem R is Levin-reducible to a search problem $R' \in \mathcal{PC}$), *then S' is $\mathcal{NP}$-complete* (resp., R' is $\mathcal{PC}$-complete).

Theorem 4.12. *Assuming $\mathcal{NP} \neq \mathcal{P}$, there exist decision problems in $\mathcal{NP} \setminus \mathcal{P}$ that are not NP-complete* (even when allowing Cook-reductions).

Promise Problems. Promise problems are natural generalizations of search and decision problems that are obtained by explicitly specifying a set of legitimate instances (rather than considering any string as a legitimate instance). See Section 5.1.

Theorem 5.5. *There exists an optimal algorithm for any candid search problem in NP*, where the candid search problem of the binary relation R consists of finding solutions whenever they exist (and behaving arbitrarily otherwise; see Definition 5.2).

Theorem 5.7. *If every set in $\mathcal{NP}$ can be Cook-reduced to some set in $\mathcal{NP} \cap \mathrm{co}\mathcal{NP}$, then $\mathcal{NP} = \mathrm{co}\mathcal{NP}$*, where $\mathrm{co}\mathcal{NP} = \{\{0,1\}^* \setminus S : S \in \mathcal{NP}\}$.