Brief Contents

Contents

Preface

Why this book?

Computational complexity theory is part of the foundation of computer science (CS). It provides fundamental insights about many areas of CS, including security, artificial intelligence, big data processing, verification, optimization, parallel computing, and distributed computing. Because computational complexity is sometimes abstract and technical, it might appear too isolated to have practical relevance. To convey how computational complexity is germane to CS as a whole, this book highlights connections to other areas of CS and emphasizes broadly useful intuitions and techniques for reasoning about computing.

Some students are reluctant to study computational complexity because it's math-heavy. This book aims to make the subject accessible to readers with limited math background. A music teacher once lamented, "They don't write easy music in hard keys." Students struggle to adapt to key signatures with lots of sharps or flats because most music written in those keys is technically demanding. For many people, mathematical rigor is a "hard key." This book provides some "easy music" to help readers acclimate and see that computational complexity is not so formidable. The appendix reviews relevant discrete math background.

Intended audience

This book is aimed at general CS graduate students, but it's also suitable for advanced undergraduates, students in related fields, working professionals, and armchair computer scientists.

The reader is assumed to have a typical undergraduate education in CS, though an intrepid reader can appreciate much of the book without such a background. The reader should have experience writing programs and a little familiarity with data structures (including stacks, queues, and dictionaries), algorithms (such as binary search and merge sort), discrete math, assembly language, and digital logic. The text sometimes alludes to basic concepts of operating systems, compilers, computer architecture, security, and artificial intelligence.

An important prerequisite is mathematical maturity: the ability to mentally connect notations to the abstract concepts they represent, to recognize valid logical reasoning, and to distinguish key ideas from technical details.

Goals of this book

The foremost reason to study computational complexity is to gain insight about the essence of computing. This book offers intuition with "proof idea" sections and clarifies definitions with plenty of concrete examples. The book sometimes only presents interesting special cases—rather than general results—when they're easier to absorb. To help the reader solidify the material, each chapter ends with exercises ranging from calisthenics to guided advanced material.

This book is "batteries included": Everything is explained from first principles, without requiring the reader to fill in details. The book is both a self-contained tutorial and a thorough reference with full proofs. The reader is advised when certain proofs are ancillary and safe to skip. A few advanced "museum exhibit" theorems are stated without proof. (Full disclosure: An auxiliary "battery" that's unfortunately not included is the existence of prime numbers and irreducible polynomials of suitable sizes. These standard theorems are a bit too deep to prove here, but we need them for constructing finite fields, so we accept them as gifts from pure math.)

This book tries to appeal to a broad CS audience by mentioning pragmatic motivations and by connecting complexity theoretic ideas to analogues elsewhere in CS. Most areas of CS benefit from computational complexity's rigorous perspective on the power and limitations of computing. The book covers subareas of complexity that form theoretical foundations of other areas of CS, such as cryptography and machine learning.

For general CS students, the book can be an accessible survey of complexity theory and provide inspiration about the benefits of rigorously formulating questions about computing. For students specializing in theoretical computer science, the book covers standard material and some advanced topics. The reader will emerge ready to study research papers in complexity.

Another goal is to rein in the necessary math background. The basic background includes logic, combinatorics, graph theory, discrete probability, and algebra limited to prime fields, polynomials, vectors, and matrices—nothing more abstract is needed. The appendix reviews notation, terminology, proof techniques, and basic lemmas. Slightly more advanced tools—including concentration bounds and power-of-2 fields—are developed in the main text. The book entirely avoids calculus, information theory, linear programming, continuous probability, and complex numbers. Inductive proofs are phrased as iterative or recursive algorithms that maintain invariants (rather than "raw induction"), befitting the CS context.

Perhaps controversially, the main text doesn't mention researchers' names that got attached to results and definitions in the literature. For example, some people may decry this book's avoidance of the name "Markov's inequality." The goals are to avoid perpetuating any misallocation of credit (since the history of CS and math is littered with attributions of questionable veracity) and to emphasize that the results and ideas are fundamental parts of reality, independent of the circumstances of their discovery. Bibliographic references and historical context (including traditional names) are relegated to the "Notes" section in each chapter. A few exceptions are when someone's name has become a common adjective with a suffix, such as "boolean" and "cartesian."

The overarching goal is to tell a story, with plotlines woven together like a satisfying novel. This book presents computational complexity as a cohesive subject whose many threads are ultimately interconnected, rather than a collection of disparate topics. Each of the book's four parts ends with a "plot twist."

Manifesto on the model of computation

Turing machines are the most traditional model of computation for teaching complexity theory. This book uses the Word RAM model instead. Advantages of Word RAM include:

- It resembles how actual computers work. Turing machines' back-and-forth sequential-access tapes may give students reservations about the relevance to real-world computing. But students generally don't need convincing that Word RAM—which is an idealized assembly language—appropriately distills the essence of computation. It's possible to define random-access Turing machines with special tapes for constructing addresses to teleport tape heads, but this is awkward and doesn't fully purge sequential access from the model.
- Implementing algorithms in the Word RAM model is reasonably straightforward (and only slightly tedious). For example, the breadth-first search algorithm naturally corresponds to a modest amount of Word RAM code. Implementing algorithms as Turing machines can be nightmarish. The author shudders to envision the state diagram of a random-access Turing machine for breadth-first search. Turing machines must fuss about determining the input size, delineating boundaries between pieces of data, doing simple operations such as addition at the bit level, and constructing memory addresses bit-by-bit. These boring details distract from the ideas and get swept under the rug ("clearly it can be done"). Word RAM helps students feel anchored because a high-level algorithm is never far from a low-level implementation.
- With Word RAM, the input size and running time conform to our intuition about algorithms. For example, consider summing N integers that fit in $O(\log N)$-bit words. With Word RAM, the input size is N words, and the running time is $O(N)$, as expected. With Turing machines, the input size is $O(N \log N)$ bits, and the running time must account for doing each addition at the bit level. Analyses of Turing machines have omnipresent polylogarithmic factors to soak up these low-level nuisances. Analyses of Word RAM implementations are consistent with our expectations about algorithms.

Disadvantages of Word RAM include:

- It's not a "theoretically minimal" model. The instruction set is chosen more for convenience than for mathematical purity. Some people find the austerity of Turing machines appealing.
- A Word RAM program must specify how the word size depends on the input size. This is extra baggage attached to a program's code. It sometimes throws a small wrench in the works, such as in the proofs of the time and space hierarchy theorems. Also, defining the set of permissible word sizes is subtle. (Some versions of Word RAM let the word size change during a computation, but the author considers that too unrealistic.)
- Combining programs that have different word sizes—or modifying programs in a way that changes the word size—requires attention to preserve the programs' intended behavior. This minor annoyance doesn't crop up with Turing machines because their "word size" is constant.

Content and organization

This book has more than enough material for two semesters. A typical one-semester graduate complexity course would cover much of the first seven chapters and selections from other chapters. The most fundamental material is in the first seven chapters. The next two chapters cover widely useful tools (pairwise uniform hashing and error correcting codes) and are independent of each

other. The rest of the chapters often use concepts from the first nine chapters but are largely independent of each other (and can be read in any order), except minor dependencies such as:

- Locally decodable codes (§10.3) are relevant to probabilistically checkable proofs (§14.4).
- Pseudorandom generators (§11.1) are relevant to encryption schemes (§12.2).
- Pseudorandom functions (§12.4) are relevant to hardness of learning circuits (§13.7) and to the natural proofs barrier (§18.7).
- Commitment schemes (§12.3) are relevant to zero-knowledge proofs (§15.4).
- Communication complexity of search problems (§17.5) is relevant to formula size complexity (§18.5).

Here are some of the author's editorial decisions about the selection and organization of material:

- The time-space lower bound for SATISFIABILITY (Chapter 4) serves as a unified excuse to introduce several fundamental concepts such as diagonalization and the polynomial hierarchy. Since the polynomial hierarchy is treated as more of a "means" than an "end," material about it is distributed across relevant chapters rather than in a dedicated chapter.
- The book avoids esoteric complexity classes and antiquated topics in structural complexity such as relativization, exact counting of witnesses, the isomorphism conjecture, and sparse decision problems.
- The chapters on concrete nonuniform models of computation are the last part of the book because the uniform model (Word RAM programs) is discarded at this point, and because the circuit complexity approach to P versus NP in the final chapter brings the book "full circle."
- When carving a sculpture, some perfectly good stone will be lost: The epilogue is a teaser for important topics that didn't make it into the text for various reasons, such as being math-heavier or harder to motivate in the broader context of CS.

Acknowledgments

My perspective on computational complexity was profoundly influenced by my advisors throughout my education—Dieter van Melkebeek, Luca Trevisan, and Toniann Pitassi—and by all my collaborators, especially Mika Göös. I thank Dieter van Melkebeek and anonymous reviewers for helpful feedback. I thank the people at Cambridge University Press for help during the publication process. The dblp website was very useful for curating references. I thank the US National Science Foundation for financial support. Finally, I thank my family for essential nontechnical support.