Mathematics and Computation

Mathematics and Computation

A Theory Revolutionizing Technology and Science

Avi Wigderson

Dedicated to the memory of my father, Pinchas Wigderson (1921–1988),
who loved people, loved puzzles, and inspired me.

Ashgabat, Turkmenistan, 1943

Contents

Acknowledgments

In this book I tried to present some of the knowledge and understanding I acquired in my four decades in the field. The main source of this knowledge was the Theory of Computation community, which has been my academic and social home throughout this period. The members of this wonderful community, especially my teachers, students, postdocs and collaborators, but also the speakers in numerous talks I attended, have been the source of this knowledge and understanding often far more than the books and journals I read. Ours is a generous and interactive community, whose members are happy to share their own knowledge and understanding with others, and are trained by the culture of the field to do so well. These interactions made (and still makes) learning a greatly joyful experience for me!

More directly, the content and presentation in this book benefited from many who carefully read earlier drafts, and responded with valuable constructive comments at all levels; this made the book much better. For this I am grateful to Scott Aaronson, Dorit Aharonov, Morteza Alimi, Noga Alon, Sanjeev Arora, Boaz Barak, Zeb Brady, Mark Braverman, Bernard Chazelle, Neil Chriss, Tom Church, Geoffroy Couteau, Dennis Dolan, Andy Drucker, Ron Fagin, Yuval Filmus, Michael Forbes, Ankit Garg, Sumegha Garg, Oded Goldreich, Renan Gross, Nadia Heninger, Gil Kalai, Vickie Kearn, Pravesh Kothari, James Lee, Alex Lubotzky, Assaf Naor, Ryan O'Donnell, Toni Pitassi, Tim Randolph, Sasha Razborov, Tim Roughgarden, Mike Saks, Peter Sarnak, Susannah Shoemaker, Amir Shpilka, Alistair Sinclair, Bill Steiger, Arpita Tripathi, Salil Vadhan, Les Valiant, Thomas Vidick, Ben Lee Volk, Cyd Westmoreland, Edna Wigderson, Yuval Wigderson, Ronald de Wolf, Amir Yehudayoff, Rich Zemel and David Zuckerman.

Very special thanks go to Sahar Batsry for many days of meticulous work on many evolving ideas I had for the cover design, to Edna and Yuval Wigderson for great help with all aspects of LaTeX, English, graphics and many other technical issues that were beyond me, and finally to Eyal and Einat Wigderson for culinary (and many other) pleasures.

I thank Princeton University Press for the production of this book, especially Nathan Carr, Vickie Kearn, Susannah Shoemaker, and above all Cyd Westmoreland, whose copyediting of this book was just amazing.

Online resources have eased many aspects of book-writing. I want to acknowledge specifically the great utility for me of Google Scholar, Wikipedia and the arXiv repository!

Some chapters in this book are revisions and extensions of material taken from my survey [Wig06], which in turn used parts of a joint survey with Goldreich in this volume [GBGL10].

Last but not least, I am grateful to Tom and Roselyne Nelsen for letting me use their beautiful home in Sun Valley, Idaho—much of this book was written in that serene environment over the past few summers.

Mathematics and Computation

1   Introduction

"$\mathcal{P}$ versus $\mathcal{NP}$ — a gift to mathematics from computer science"

–Steve Smale

Here is just one tip of the iceberg we'll explore in this book: How much time does it take to find the prime factors of a 1,000-digit integer? The facts are that (1) we can't even roughly estimate the answer: it could be less than a second or more than a million years, and (2) practically all electronic commerce and Internet security systems in existence today rest on the belief that it takes more than a million years!

Digesting even this single example begins to illuminate the conceptual revolution of *computational complexity theory*, to which this book is devoted. It illustrates how a pure number theoretic problem, which has been studied for millennia by mathematicians, becomes a cornerstone of a trillion-dollar industry, on which practically all people, companies, and countries crucially depend. Extracting this novel meaning and utility relies on making the above problem precise, and on transforming the informal statement of (2) into a mathematical theorem. This in turn requires formal definitions of such concepts as *algorithm, efficiency, secret*, and *randomness*, among others, including several new notions of *proof*. The difficulty of resolving (the now all-important) challenge (1), namely, proving the hardness of factoring or suggesting alternatives to it, is intimately related to the great conundrum of $\mathcal{P}$ vs. $\mathcal{NP}$. And as a final twist in this plot, a fork appeared in our computational path, by which the answer to (1) may radically depend on whether we allow classical or quantum physics to power our computers. This new possibility has propelled huge investments in academia and industry to attempt to physically realize the potential of quantum computers. It also demands revisiting and redefining the very concepts mentioned above, as well as many physical ones like *entanglement* and *decoherence*, interacting with quantum mechanics and proposing novel ways of testing its foundations.

The book you are reading will explore the mathematical and intellectual aspects of this goldmine. It will explain *computational complexity theory*, the concepts this theory created and revolutionized, and its many connections and interactions with mathematics. In its half-century of existence, computational complexity theory has developed into a rich, deep, and broad mathematical theory with remarkable achievements and formidable challenges. It has forged strong connections with most other mathematical fields and at the same time is having a major practical impact on the *technological revolution* affecting all aspects of our society (of which Internet security and quantum computing above are "mere" examples).

Computational complexity theory is a central subfield of the *theory of computation* (ToC), and is playing a pivotal role in its evolution. This theory stands with the great ones of physics, biology, math, and economics, and is central to a new *scientific revolution* informed by computation. I have devoted the final chapter of this book (which can be read first) to a panoramic overview of ToC. That chapter describes the intellectual supernova that the theory of computing has created and continues to shape. It reveals the broad reach of ToC to all sciences, technology, and society, and discusses its methodology, challenges, and unique position in the intellectual sphere.

Below I review the long history of the interactions of computation and mathematics. I proceed with a short overview of the evolution and nature of computational complexity. I then describe the structure, scope, and the intended audience of this book, followed by a chapter-by-chapter description of its contents.

1.1   On the interactions of math and computation

The Theory of Computation is the study of the formal foundations of computer science and technology. This dynamic and rapidly expanding field straddles mathematics and computer science. It has benefited tremendously from the very different characters, motivations, and traditions of these parent disciplines. Both sides naturally emerged from its birth, in the "big bang" of Turing's seminal 1936 paper [Tur36], "On computable numbers, with an application to the Entscheidungsproblem." This is a paper written by a PhD student, in the area of mathematical logic, which, combined with its long title, might seem to condemn it to obscurity. However, with Turing's incredible clarity of vision, exposition, and motivation, it is an inspiring model in mathematical modeling with singular impact. This paper formally defined an *algorithm* in the form of what we call today the "Turing machine." On one hand, the Turing machine is a formal

mathematical model of computation, enabling for the first time the rigorous definition of computational tasks, the algorithms to solve them, and the basic resources these require (in particular, allowing Turing to prove that very basic tasks are uncomputable). On the other hand, the extremely elegant definition of the Turing machine allowed its simple, logical design to be readily implemented in hardware and software, igniting the computer revolution.

These theoretical and practical aspects form the dual nature of ToC and strongly influenced the field and its evolution. On the mathematical side, the abstract notion of computation revealed itself as an extremely deep and mysterious notion that illuminates other, often well-studied concepts in a new light. In pursuing the abstract study of computation, ToC progresses like any other mathematical field. Its researchers prove theorems and follow mathematical culture to generalize, simplify, and create variations, following their instincts based on esthetics and beauty. On the practical side, the universal applicability of automated computation fueled the rapid development of computer technology, which now dominates our life. The interaction between theory and practice never stops. The evolving world of computer science and industry continuously creates new types and properties of computation, which need theoretical modeling and understanding, and directly impacts the mathematical evolution of ToC. Conversely, the ideas, models, and techniques generated there feed back into the practical world. Besides technology, more recent and growing sources of external influence on ToC are nature and science. Many natural processes can (and should) be understood as information processes and demand similar computational understanding. Here again theoretical modeling, techniques, and new theoretical questions feed back to suggest experiments and better understanding of scientific data. Much more on these connections is discussed in Chapter 20.

Needless to say, mathematics and computation did not meet in 1936 for the first time; they have been tied to each other from the dawn of human civilization. Indeed, ancient mathematics developed primarily from the need to compute, be it for predicting natural phenomena of all types, managing crops and livestock, manufacturing and building, trading commodities, and planning for the future. Devising representations of numbers and developing efficient methods for performing arithmetic on them were thus central. More generally, in a very fundamental way, a mathematical understanding could solve any practical problem only *through* a computational process applied to the data at hand. So, even though algorithms were formally defined only in the twentieth century, mathematicians and scientists continuously devised, described, and used better and better algorithms (albeit informally explained and rarely analyzed) for the computations required to draw conclusions from their theories. Examples abound, and we list just a few highlights. Euclid, working ca. 300 BCE, devised his fast GCD[1] algorithm to bypass the need to laboriously factor integers when simplifying fractions. Euclid's famous 13-volume *Elements*, the central math text for many centuries, contains dozens of other algorithms to compute numerical and geometric quantities and structures. In the same era, Chinese mathematicians compiled *The Nine Chapters on the Mathematical Art*, which contains many computational methods, including "Gaussian elimination" (for solving systems of linear equations). In the ninth century, al-Khwārizmī (after whom "algorithm" is named) wrote his books *Compendious Book on Calculation by Completion and Balancing* and *On the Hindu Art of Reckoning*. These books respectively expound on everything known up to that time about algorithms for algebraic and arithmetic problems, such as solving quadratic equations and linear systems, and performing arithmetic operations in the decimal system. The very reason that the decimal system survived as the dominant way to represent numbers is the usefulness of these efficient algorithms for performing arithmetic on arbitrarily large numbers so represented.

The "modern era" has intensified these connections between math and computation. Again, examples. During the Renaissance, mathematicians found *formulas*, the most basic computational recipe, for solving cubic and quartic equations via radicals.[2] Indeed, famous competitions between Tartaglia, Piore, Ferrari, and others in the early 1500s were all about who had a faster algorithm for solving cubic equations. The Abel-Ruffini theorem that the quintic equation has no such formula is perhaps the earliest *hardness result*: It proves the non existence of an algorithm for a concrete problem in a precise computational model. Newton's *Principia Mathematica* is a masterpiece not only of grand scientific and mathematical theories; it is also a masterpiece of algorithms for computing the predictions of these theories. Perhaps the most famous and

---

[1]The GCD (greatest common divisor) problem is to compute the largest integer that evenly divides two other integers.

[2]Namely, using arithmetic operations and taking roots.

most general is "Newton's method" for approximating the roots of real polynomials of arbitrary degree (practically bypassing the Abel-Ruffini obstacle mentioned above). The same can be said about Gauss' magnum opus, *Disquisitiones Arithmeticae*—it is full of algorithms and computational methods. One famous example (published after his death), is his discovery[3] of the fast Fourier transform (FFT), the central algorithm of signal processing, some 150 years before its "official" discovery by J. W. Cooley and J. W. Tukey. Aiming beyond concrete problems, Leibniz, Babbage, Lovelace, and others pioneered explicit attempts to design, build, and program general-purpose computational devices. Finally, Hilbert dreamed of resting all of mathematics on computational foundations, seeking a "mechanical procedure" that would (in principle) determine all mathematical truths. He believed that truth and proof coincide (i.e., that every true statement has a valid proof), and that such proofs can be found automatically by such a computational procedure. The quest to formalize Hilbert's program in mathematical logic led directly to the works of Gödel, Church, Post, and Turing. Their work shattered Hilbert's dreams (proving them unattainable), but in doing so gave birth to formal definitions of computation and algorithms. Once these theoretical foundations were laid, the computer revolution commenced.

The birth of computer science, and with it ToC steadily enhanced, deepened, and diversified the interactions between mathematics and computation, which in the past few decades have been growing explosively. These interactions can be roughly divided into four (naturally overlapping) categories. The first two categories arise from one field using the expertise developed by the other; these interactions are often mainly one directional. The next two categories are more complex and are highly interactive. We will see many of these interactions at play throughout the book.

- One type of interaction arises from the need of ToC to use general mathematical techniques and results. Initially these sources were restricted to areas having a natural affinity with computer science, like logic and discrete mathematics. However, as ToC developed, becoming deeper and broader, it needed to import techniques and results from diverse mathematical areas, some quite unexpected. Such examples include the use of analytic and geometric techniques for understanding approximation algorithms, the use of topological methods for studying distributed systems, and the use of number theory and algebraic geometry in the construction of pseudo-random objects.

- The opposite type of interaction is the need of mathematics to use algorithms (and computers). As mentioned, mathematicians needed algorithms and had developed them for centuries. But after Turing, ToC transformed algorithm design into a comprehensive theory, ready to be used and applied. This theory has general techniques for maintaining and manipulating information of all types and methods for comparing the qualities and analyzing the resources of these algorithms. At the same time, computers became available. This confluence gave rise to a huge boom in developing specific algorithms and software for mathematicians in almost every field, with libraries of computational tools for algebra, topology, group theory, geometry, statistics, and other fields. On a different front, there is growing development and use of computer programs for mathematical proof verification as for well as for proof discovery.

- A deeper, more fundamental source of interaction is the vast number of mathematical theorems that guarantee the existence of some mathematical object. It is by now a reflexive reaction to wonder: *Can the object guaranteed to exist be efficiently found?* In many cases, some mentioned above, there are good practical reasons to seek such procedures. On a more philosophical level, nonconstructive existence proofs (like Hilbert's first proof of the finite basis theorem and Cantor's proof that most real numbers are not algebraic) appalled members of the mathematics community. Even in finite settings, existence proofs beg better understanding despite the availability of brute-force (but highly inefficient) search algorithms for the required object. We have mounting evidence in diverse areas of math that even without any direct, practical need or philosophical desire for such efficient algorithms, seeking them anyway invariably leads to a deeper understanding of the mathematical field at hand. This quest raises new questions and uncovers new structures, sometimes reviving "well-understood" subjects.

---

[3]For the purpose of efficiently predicting the orbits of certain asteroids.

- The final source of interaction arises because, perhaps surprisingly, the study of computation leads to the production of new mathematical results, theories, and problems, which are not algorithmic but instead structural in nature. These naturally arise from the need both to analyze algorithms and to prove hardness results. Thus were born completely new probabilistic concentration results, geometric incidence theorems, combinatorial regularity lemmas, isoperimetric inequalities, algebraic identities, statistical tests, and more. These results inspired collaborations across many mathematical areas, which in some cases are already quite well established and in others are just now budding.

Mathematics and computation are linked by numerous strong bonds, some far beyond the scope of this book. We will witness here these mutual interactions and interests in the field of computational complexity.

1.2   Computational complexity theory

The early decades of research on ToC centered on understanding which computational problems can and which cannot be solved by algorithms. But it quickly became evident that this dividing line is too coarse; many problems that can be solved in principle remain intractable, as the best algorithms to solve them will terminate long after anyone cared or lived to see the answer. This created a need for a much more refined theory that will account for the *performance* of different algorithms. Thus was born computational complexity theory in the 1960s, with the initial charge of understanding *efficient computation* in the most general sense: *determining the minimal amounts of natural resources (like time, memory, and communication) needed to solve natural computational tasks by natural computational models*. Responding to this challenge, the theory developed a powerful toolkit of algorithmic techniques and ways to analyze them, as well as a classification system of computational problems that uses *complexity classes*. These results also led to the formulation of natural long-term goals for the field regarding the power and limits of efficient computation. But these developments turned out to be only half of the story, as the resources consumed by algorithms are only the very basic properties of computation, and the utility of algorithms in different contexts presented other properties to explore.

With time, and with such a variety of internal and external motivations, computational complexity theory has greatly expanded its goals. It took on the computational modeling and the understanding of a variety of central notions, some studied for centuries by great minds, including the concepts of *secret, proof, learning, knowledge, randomness, interaction, evolution, game, strategy, coordination,* and *synchrony*, among others. This computational lens often resulted in completely new meanings for these old concepts.[4] Moreover, in some cases, the resulting theories predated (and indeed enabled) significant technological advances.[5] In other cases, these theories formed the basis of interactions with other sciences.

Thus, starting with the goal of understanding what can be efficiently *computed*, a host of natural long-term goals of deep conceptual meaning emerged. What can be efficiently learned? What can be efficiently proved? Is verifying a proof much easier than finding one? What does a machine know? What is the power of randomness in algorithms? Can we effectively use natural sources of randomness? What is the power of quantum mechanical computers? Can we utilize quantum phenomena in algorithms? In settings where different computational entities have different (possibly conflicting) incentives, what can be achieved jointly? Privately? Can computers efficiently simulate nature, or the brain?

The study of efficient computation has created a powerful methodology with which to investigate such questions. Here are some of its important principles, which we will see in action repeatedly, and whose meaning will become clearer as we proceed in this book. *Computational modeling*: uncover the underlying basic operations, information flow, and resources of processes. *Efficiency*: attempt to minimize resources used and their trade-offs. *Asymptotic thinking*: study problems on larger and larger objects, as structure

---

[4]For example, this approach suggests that in proofs, one can decouple verification and understanding: *every* mathematical theorem can be proved in a convincing manner that nonetheless reveals absolutely no information except its validity (This will be discussed in Section 10.2.)

[5]The best example is cryptography, which in the 1980s was purely motivated by a collection of fun intellectual challenges, like playing poker over the telephone, but developed into a theory that enabled the explosive growth of the Internet and e-commerce. (This will be discussed in Chapter 18.)

often reveals itself in the limit. *Adversarial thinking*: always prepare for the worst, replacing specific and structural restrictions by general, computational ones—such more stringent demands often make things simpler to understand! *Classification*: organize problems into (complexity) classes according to the resources they require. *Reductions*: ignore ignorance, and even if you can't efficiently solve a problem, assume that you can, and explore which other problems it would help solve efficiently. *Completeness*: identify the most difficult problems in a complexity class.[6] *Barriers*: when stuck for a long time on a major question, abstract all known techniques used for it so far and try to formally argue that they will not suffice for its resolution.

These principles work extremely well with one another, and in surprisingly diverse settings, especially when applied at the right level of abstraction (which I believe has been indeed cleverly chosen, repeatedly). This methodology allowed ToC to uncover hidden connections among different fields and create a beautiful edifice of structure, a remarkable order in a vast collection of notions, problems, models, resources, and motivations. While many of these principles are in use throughout mathematics and science, I believe that the disciplined, systematic use that has become ingrained in the culture of computational complexity—especially of problem classification via (appropriate) reductions and completeness—has great potential to further enhance other academic fields.

The field of Computational complexity is extremely active and dynamic. While I have attempted to describe, besides the fundamentals, some of the recent advances in most areas, I expect the state of the art will continue to expand quickly. Thus, some of the open problems will become theorems, new research directions will be created, and new open problems will emerge. Indeed, this has happened repeatedly in the few years it has taken me to write this book.

1.3   The nature, purpose, and style of this book

Computational complexity theory has been my intellectual (and social) home for almost 40 years. During this period, I have written survey articles and have given many more survey lectures on various aspects of of this field. This book grew out of these expositions and out of the many other aspects I had planned to write and talk about, but had never gotten around to. Indeed, breadth of scope is an important goal of the book. Furthermore, as in my lectures and surveys, so in this book, I try to explain not only what we do, but also why we do it, why is it so important, and why is it so much fun!

The book explores the foundations and some of the main research directions of computational complexity theory, and their many interactions with other branches of mathematics. The diversity of these computational settings is revealed when we discuss the contents of every chapter below. For every research area, the book focuses on the main aspects of computation it attempts to *model*. The book presents the notions, goals, results, and open problems for each area in turn, all from a conceptual perspective, providing ample motivation and intuition. It describes the history and evolution of ideas leading to different notions and results and explains their meaning and utility. It also highlights the rich tapestry of (often surprising and unexpected) connections among the different subareas of computational complexity theory; this unity of the field is an important part of the field's success.

To highlight the conceptual perspective, the material is generally presented at a high level and in somewhat informal fashion. Almost no proofs are given, and I focus on discussions of general proof techniques and key ideas at an informal level. Precise definitions, theorem statements, and of course detailed proofs for many topics discussed can be found in the excellent textbooks on computational complexity [Pap03, Gol08, AB09, MM11]. Also, for historical reasons and greater detail, I provide many references to original papers, more specialized textbooks, and survey articles in every chapter.

1.4   Who is this book for?

I view this book as useful to several audiences in several different ways.

- First, it is an invitation to advanced undergraduates and beginning graduate students in math, computer science, and related fields, to find out what this field is about, get excited about it, and join it

---

[6]Namely, those which all other problems in the class reduce to in the sense mentioned above.

as researchers.

- Second, it should serve graduate students and young researchers working in some area of CS theory to broaden their views and deepen their understanding about other parts of the field and their interconnectivity.

- Third, computer scientists, mathematicians, researchers from other fields, and motivated nonacademics can get a high-level view of computational complexity, its broad scope, achievements, and ambitions.

- Last but not least, educators in the field can use different parts of the book for planning and supplementing a variety of undergraduate and graduate courses. I hope that the conceptual view of the field, its methodology, its unity, and the beauty and excitement of its achievements and challenges that I have labored to present here will help inform and animate these courses.

Different chapters may require somewhat different backgrounds, but the introduction to each aims to be welcoming and gentle. The first two chapters and the last one should be accessible to most of the above mentioned audiences.

Let me conclude this section with a piece of advice that may be useful to some readers. It can be tempting to read this book quickly. Many parts of the book require hardly any specific prior knowledge and rely mostly on the mathematical maturity needed to take in the definitions and notions introduced. Hopefully, the story telling makes the reading even easier. However, the book (like the field itself) is conceptually dense, and in some parts the concentration of concepts and ideas requires, I believe, slowing down, rereading, and possibly looking at a relevant reference to clarify and solidify the material and its meaning in your mind.

1.5   Organization of the book

Below I summarize the contents of each chapter in the book. Naturally, some of the notions mentioned below will only be explained in the chapters themselves. After the introductory Chapters 2 and 3, the remaining ones can be read in almost any order. Central concepts (besides computation itself) that sweep across several chapters include *randomness* (Chapters 7–10), *proof* (Chapters 3, 6, and 10) and *hardness* (Chapters 5, 6, and 12).

Different partitions can be made around the focus of different sets of chapters. Chapters 2–12 focus mostly on one computational resource, *time*, namely, the number of steps taken by a single machine (of various types) to solve a problem. Chapters 14–19 (as well as 10) deal with other resources and with more complex computational environments, in which interactions take place among several computational devices. Finally, while mathematical *modeling* is an important part of almost every chapter, it is even more so for the complex computational environments we'll meet in Chapters 15–19, where modeling options, choices, and rationales are discussed at greater length. Chapters 13 and 20 are standalone surveys, the first on concrete interactions between math and computational complexity and the second on the Theory of Computation. Here are brief descriptions of each chapter (the headlines below may differ from the chapter title).

Prelude: computation and mathematical understanding
Chapter 2 is the prelude to the arrival of computational complexity, starting with the formalization of the notion of *algorithm* as the Turing machine. We discuss basic computational problems in mathematics and algorithms for them. We then explore the relevance of the boundary between decidable and undecidable problems about classes of mathematical structures, in the hope of completely understanding them.

Computational complexity 101 and the $\mathcal{P}$ vs. $\mathcal{NP}$ question
Chapter 3 introduces the basic concepts of computational complexity: decision problems, time complexity, polynomial vs. exponential time, efficient algorithms, and the class $\mathcal{P}$. We define efficient verification and the class $\mathcal{NP}$, the first computational notion of proof. We proceed with efficient reductions between problems and the notion of completeness. We then introduce $\mathcal{NP}$-complete problems and the $\mathcal{P}$ vs. $\mathcal{NP}$ question. Finally, we discuss related problems and complexity classes. For all these notions, I motivate some of the

choices made and explain their importance for computer science, math, and beyond.

Different types of computational problems and complexity classes

Chapter 4 introduces new types of *questions* one can ask about an input beyond classification, including counting, approximation, and search. We also move from worst-case performance of algorithms to success or failure on average, and discuss the related notions of one-way and trap-door functions underlying cryptography. I explain how the methodology developed in the previous chapter leads to other complexity classes, reductions, and completeness. This begins to paint the richer structure organizing problems above, below, and "around" $\mathcal{NP}$.

Hardness and the difficulties it presents

Chapter 5 discusses *lower bounds*—the major challenge of proving that $\mathcal{P}$ is different from $\mathcal{NP}$, and more generally, proving that some natural computational problems are hard. Central to this section is the model of Boolean circuits, a "hardware" analog of Turing machines. We review the main techniques used for lower bounds on restricted forms of circuits and Turing machines. We also discuss the introspective barrier results, explaining why these techniques seem to fall short of the real thing: lower bounds for general models.

How deep is your proof?

Chapter 6 introduces *proof complexity*, another view of the basic concept of *proof*. Proof complexity applies computational complexity methodology to quantify the difficulty of proving natural theorems. We describe a variety of propositional proof systems—geometric, algebraic, and logical—all capturing different ways and intuitions of making deductions for proving natural tautologies. I explain the ties among proof systems, algorithms, circuit complexity, and the *space: the final frontier* problem. We review the main results and challenges in proving lower bounds in this setting.

The power and weakness of randomness for algorithms

Chapter 7 reviews using randomness to enhance the power of algorithms. We define probabilistic algorithms and the class $\mathcal{BPP}$ of problems they solve efficiently. We discuss such problems (among numerous others) for which no fast deterministic algorithms are known, which suggests that randomness is powerful. However, this intuition may be an illusion. We next introduce the fundamental notions of *computational pseudo-randomness, pseudo-random generators*, the *hardness vs. randomness paradigm*, and *de-randomization*. These ideas suggest that randomness is weak, at least assuming hardness statements like $\mathcal{P} \neq \mathcal{NP}$. The chapter concludes with a discussion of the evolution and sources of these ideas, their surprising consequences, and their impact beyond the power of randomness.

Is π random?

Chapter 8 covers "random looking" deterministic structures. We discuss "abstract" pseudo-randomness, a general framework that extends computational pseudo-randomness and accommodates a variety of natural problems in mathematics and computer science. We define pseudo-random properties and discuss the question of deterministically finding pseudo-random objects. We will see how the $\mathcal{P}$ vs. $\mathcal{NP}$ problem, the Riemann hypothesis, and many other problems that naturally fall into this framework can be viewed as questions about pseudo-randomness. Finally, we discuss the structure vs. pseudo-randomness dichotomy, a paradigm for proving theorems in a variety of areas, and examine the scope of this idea.

Utilizing the unpredictability of the weather, stock prices, quantum effects, etc.

Chapter 9 discusses weak random sources, a mathematical model of some natural phenomena that seem to be somewhat unpredictable but may be far from a perfect stream of random bits. We raise the question of if and how probabilistic algorithms can use such weak randomness and define the main object used to answer this question—the randomness extractor. We then explore the evolution of ideas leading to effi-

cient constructions of extractors, and the remarkable utility of this pseudo-random object for other purposes.

Interactive proofs: teaching students with coin tosses

Chapter 10 addresses yet again, proofs this time concerning the impact of introducing randomness and interaction into the definition of proofs. These new notions of proofs give rise to new complexity classes, like $\mathcal{IP}$ and $\mathcal{PCP}$, and their surprising characterization in terms of standard complexity classes. We explore how this setting allows new types of proofs, like zero-knowledge proofs and spot-checking proofs, and their implications for cryptography and hardness-of-approximation.

Schrödinger's laptop: algorithms meet quantum mechanics

Chapter 11 introduces quantum computing: algorithms endowed with the ability to use quantum mechanical effects in their computation. We discuss important algorithms for this theoretical model, how they motivated a large-scale effort to build quantum computers, and the status of this effort. We extend the idea of proofs again by using this notion and discuss complete problems for quantum proofs. This turns out to connect directly to quantum Hamiltonian dynamics, a central area in condensed matter physics, and we explore some of the interactions between the fields. Finally, we discuss the power of interactive proofs to answer the basic question: Is quantum mechanics falsifiable?

Arithmetic complexity: plus and times revisited

Chapter 12 leaves the Boolean domain and introduces the model of arithmetic circuits, which use arithmetic operations to compute polynomials over (large) fields. We review the main results and open problems in this area, relating it to the study of Boolean circuits. We exposit Valiant's arithmetic complexity theory, its main complexity classes $\mathcal{VP}$ and $\mathcal{VNP}$, complete problems, the determinant, and permanent polynomials. We discuss a recent approach to solve the $\mathcal{VP}$ vs. $\mathcal{VNP}$ problem via algebraic geometry and representation theory. We also survey a collection of restricted models for which strong lower bounds are known.

All the chapters so far focus on one primary computational resource: *time* (or more generally, the number of elementary operations). Before broadening our scope, we take a break with an interlude.

Interlude: vignettes of interaction between math and computation

Chapter 13 is different from the others. It is the middle of the book, and we take a (technical) break. In this interlude, we present a collection of short surveys on the interactions of complexity theorists and mathematicians in different mathematical areas. These vignettes span a wide spectrum of topics, as well as different types of interactions and motivations for them. They reveal the breadth of interests of computational complexity and exemplify the "computational lens" on mathematics.

After this breather, we move to study a larger variety of computational problems, resources, models, and properties. Modeling complex computational situations becomes even more important, interesting, and difficult, as (among other things) the number of participants in computation increases, the full input (or even the problem being solved) may not be known, and new efficiency measures and success criteria enter.

Space complexity: memory bottlenecks

Up to this point, the main complexity measure we have studied is *time*, or more generally, the number of basic operations performed by an algorithm. **Chapter 14** discusses *space* complexity: the memory requirements of algorithms. After introducing some basic results and open problems of this area, we focus on two specific issues in which surprising feats can be performed with surprisingly little memory. We first discuss the streaming model and the sketching technique. We then demonstrate how to count to arbitrarily high numbers with only constant memory.

Communication complexity: information bottlenecks and noise

Chapter 15 introduces communication complexity—an extremely simple, completely information theoretic model for two-party communication. Its study, however, reveals surprising depth and breadth, and basic results in this simple model turn out to have important applications for understanding a surprisingly diverse set of computational models. We also discuss how this *interactive* communication setting suggests extensions of classical questions in the fields of information theory and coding theory. We review some of the main results and challenges in understanding these extensions.

On-line computation: Is clairvoyance overrated?

Chapter 16 discusses *on-line* algorithms—reactive systems that need to respond to a continuous stream of requests or signals, attempting to optimize a goal that depends on the *future*. We first explore how to model and measure the quality of such algorithms via the notion of *competitive analysis*. We then see once again the surprising power such restricted algorithms can have in a variety of situations, from operating systems to the stock market.

Learning: How do programs recognize spam, know your tastes, and beat you at chess?

Chapter 17 discusses computational learning theory and the complex task of understanding and designing systems that make sense and cope with and thrive in unfamiliar and unspecified environments. We examine fundamental modeling issues and proposals in the teacher-student framework of *supervised learning*, aimed at capturing the ability to generalize given examples to form concepts. We review two different approaches to modeling learning in this setting, the "logical" approach of "identification in the limit" and the "statistical" approach of "distribution-free learning." We look at some learning algorithms in these settings, their key methods and insights into their analysis, and the severe limitations of both approaches.

Crypto: secrets and lies, knowledge and trust

Chapter 18 covers cryptography, one of the most complex computational environments possible. What are the guts of Internet purchasing or electronic voting protocols? Here many agents and their algorithms interact to achieve some common goal, necessarily using agents' personal data. However, these agents wish to keep the information private from illegitimate eavesdroppers, and ensure that achieving their goals is resilient to saboteurs. We explore the wealth of cryptographic tasks and constraints, and the principles of modeling them mathematically. We'll see how numerous, provably impossible tasks (like playing a game of poker over the telephone) paradoxically *can* be performed when computational power is limited. We discuss the evolution of the comprehensive theory that enables achieving these tasks.

Distributed environments: asynchrony and symmetry breaking

Chapter 19, on distributed computation, also deals with interacting parties. However, our focus will be on a very different hurdle: *asynchrony*. In this case, participants have no common clock and must compute in the presence of arbitrary communication delays. We discuss modeling such constraints and the limitations they impose. Here, too, a beautiful theory evolved that charts precisely the boundary between which tasks are possible and which are not. Part of this theory rests on deep connections that these questions have to topology.

Epilogue: the nature of ToC

Chapter 20 concludes this book with a panoramic view of ToC and its immense intellectual impact. It aims to survey many aspects of the field, both scientific and social. We will survey some of the explosion of connections of ToC with the natural and social sciences, integrating the computational lens into models and theories of nature and society. ToC is revealed as an independent academic discipline, central to most others. We discuss some of the field's intellectual and educational long-term challenges, its social character, and its adaptation to the growing role it will have to face as its scope and mission expand.

1.6   Notation and conventions

Most notation in the book is introduced as we need it, and we include here only the important asymptotic notation. We also list some conventions and tips.

- **Undefined notions** When a notion unfamiliar to you is mentioned but not immediately defined, then it is not crucial to understand its meaning for comprehending the text (but feel free to look it up, e.g., on Wikipedia).

- **References** When a list of references is given, it will typically be chronological. This may not always be evident from the publication year, due mainly to the fact that I was trying always to supply the most comprehensive and up to date version. In some cases these are journal papers (which take longest to appear), some are conference publications (which appear faster), and in rare cases only electronic versions exist (which are instantaneous). Also, when I call a result "recent", please note that this is relative to the publication date of the book.

- **Footnotes** There are *many* footnotes in this book. In almost all cases, they are designed to enrich, elaborate or further explain something. However, the text itself should be self-contained without them. Therefore you can safely skip footnotes; this will rarely affect understanding.

- **iff** The shorthand *iff* will be used throughout to mean "if and only if".

- **Asymptotic notation** Crucial to most chapters is the following asymptotic notation, relating "growth in the limit" of functions on the integers. Let f, g be two integer functions. Then we denote:

  * $f = O(g)$, if for some positive constant C, for all large enough n, $f(n) \leq C \cdot g(n)$.
  * $f = \Omega(g)$, if for some positive constant c, for all large enough n, $f(n) \geq c \cdot g(n)$.
  * $f = \Theta(g)$, if both $f = O(g)$ and $f = \Omega(g)$ hold.
  * $f = o(g)$, if $f(n)/g(n)$ tends to zero as n tends to infinity.

We say that an integer function f grows (at most) *polynomially* if there are constants A, c such that for all n, $f(n) \leq An^c$.
We say that f grows (at most) *exponentially* if there are constants A, c such that for all n, $f(n) \leq A \exp(n^c)$.

2   *Prelude*: Computation, undecidability, and limits to mathematical knowledge

This short section will briefly recount the history of ideas leading the the birth of computational complexity theory.

Mathematical classification problems   Which mathematical structures can we hope to understand? Consider any particular class of mathematical objects and any particular relevant property. We seek to *understand* which of the objects have the property and which do not. Examples of this very general *classification problem* include the following[1]:

1. Which Diophantine equations have solutions?

2. Which knots are unknotted? (see Figure 1)

3. Which planar maps are 4-colorable? (see Figure 2)

4. Which theorems are provable in Peano arithmetic?

5. Which pairs of smooth manifolds are diffeomorphic?

6. Which elementary statements about the reals are true?

7. Which elliptic curves are modular?

8. Which dynamical systems are chaotic?

NO   YES
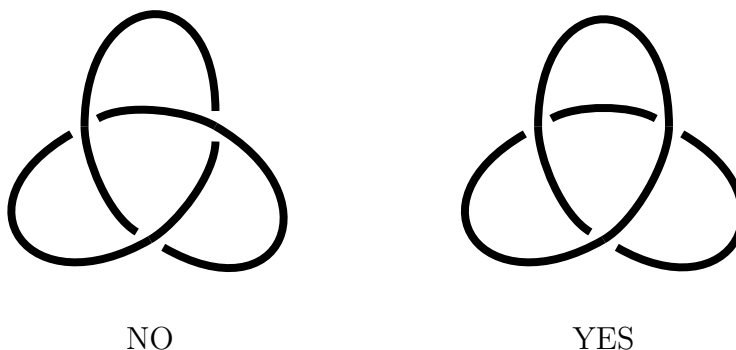
Figure 1. Instances of problem 2 and their classification. The left is a diagram of the Trefoil knot and the right is one of the Unknot.

Understanding and algorithms   A central question is what we mean by *understanding*. When are we satisfied that our classification problem has been reasonably solved? Are there problems like these that we can never solve? A central observation (popularized mainly by David Hilbert) is that "satisfactory" solutions usually provide (explicitly or implicitly) *mechanical procedures*, which when applied to an object, determine (in finite time) whether it has the property. Hilbert's problems 1 and 4 above were stated, it seems, with the expectation that the answer would be positive; namely, that mathematicians would be able to understand them in this very computational sense.

---

[1]It is not essential that you understand every mathematical notion mentioned below. If curious, reading a Wikipedia-level page should be more than enough for our purposes here.

YES   YES
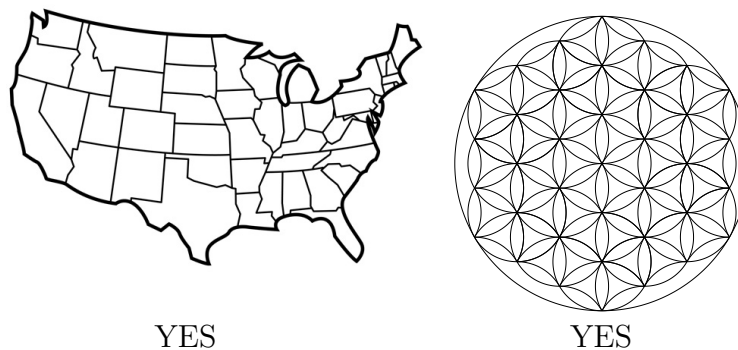
Figure 2. Instances of problem 3 and their classification. Both maps are 4-colorable.

So, Hilbert identified mathematical knowledge with computational access to answers, but never formally defined computation. This task was taken up by logicians in the early twentieth century and was met with resounding success in the 1930s. The breakthrough developments by Gödel, Turing, Church, and others led to several quite different formal definitions of computation, which happily turned out to be identical in power. Of all, Turing's 1936 paper [Tur36] was most influential. *Indeed, it is easily the most influential math paper in history.* We already mentioned in Section 1.1 that in this extremely readable paper, Turing gave birth to the discipline of computer science and ignited the computer revolution, which radically transformed society. Turing's model of computation (quickly named a *Turing machine*) became one of the greatest intellectual inventions ever. Its elegant and simple design on the one hand, and its universal power (elucidated and exemplified by Turing) on the other immediately led to implementations, and the rapid progress since then has forever changed life on Earth. This paper serves as one of the most powerful demonstrations of how excellent theory predates and enables remarkable technological and scientific advances. But on top of all these, Turing's paper also resolved problems 1 and 4 above! In the negative!! Let us see how.

With the Turing machine, we finally have a rigorous definition of computation, giving formal meaning to Hilbert's questions. It allows proving mathematical theorems about what "mechanical procedures" can and cannot do! *Turing defined an* **algorithm** *(also called a* decision procedure*) to be a Turing machine (in modern language, simply a computer program) that halts on every input in finite time.* So, algorithms compute functions, and being finite objects themselves, one immediately sees from a Cantor-like diagonalization argument that some (indeed almost all) functions are *not* computable by algorithms. Such functions are also called *undecidable*; they have no decision procedure. But Turing went further and showed that specific, natural functions, like Hilbert's Entscheidungsproblem (4) above were undecidable (this was independently proved by Church as well). Turing's elegant 1-page proof adapts a Gödelian self-reference argument on a Turing machine.[2] These demonstrate powerfully the mathematical value of Turing's basic computational model.

Decidability and undecidability   Turing thus shattered Hilbert's first dream. Problem 4 being undecidable means that we will never understand in general, in Hilbert's sense, which theorems are provable (say, in Peano arithmetic): No algorithm can discern provable from unprovable theorems. It took 35 more years to do the same to Hilbert's problem 1. Its undecidability (proved by Davis, Putnam, Robinson, and Mattiasevich in 1970) says that we will never understand in this way polynomial equations over integers: No algorithm can discern solvable from unsolvable ones.

A crucial ingredient in those (and all other undecidability) results is showing that each of these mathematical structures (Peano proofs, integer polynomials) can *"encode computation"* (in particular, these seemingly

---

[2]As a side bonus, a similar argument gives a short proof of Gödel's incompleteness theorem, a fact which for some reason is still hidden from many undergraduates taking logic courses.

static objects encode a dynamic process). This is known today to hold for many different mathematical structures in algebra, topology, geometry, analysis, logic, and more, even though *a priori* the structures studied seem to be completely unrelated to computation. This ubiquity makes every mathematician a potential computer scientist in disguise. We shall return to refined versions of this idea later.

Naturally, such negative results did not stop mathematical work on these structures and properties—they merely suggested the study of interesting subclasses of the given objects. Specific classes of Diophantine equations were understood much better, for example, Fermat's Last Theorem and the resolution of problem (7) regarding the modularity of elliptic curves. The same holds for restricted logics for number theory (e.g. Presburger arithmetic).

The notion of a decision procedure (or algorithm) as a minimal requirement for understanding a mathematical problem has also led to direct positive results. It suggests that we should look for a decision procedure as *a means*, or as the *first step* for understanding a problem. With this goal in mind, Haken [Hak61] showed how knots can be understood in this sense, designing a decision procedure for problem (2), determining knottedness. Similarly Tarski [Tar51] showed that closed real fields can be thus understood, designing a decision procedure for problem (6). Naturally, significant *mathematical, structural* understanding was needed to develop these algorithms. Haken developed the theory of *normal surfaces* and Tarski invented *quantifier elimination* for their algorithms; in both cases, these ideas and techniques became cornerstones of their respective fields.

These important examples, and many others like them, only underscore what has been obvious for centuries: mathematical and algorithmic understanding are strongly related and often go hand in hand, as discussed at length in the introduction. And what was true in previous centuries is still true in this one: The language of algorithms is compatible with and actually generalizes the language of equations and formulas (which are special cases of algorithms), and is a powerful language for understanding and explaining complex mathematical structures.

The many decision procedures developed for basic mathematical classification problems, such as Haken's and Tarski's solutions to problems (2) and (6), respectively, demonstrate this notion of algorithmic understanding *in principle*. After all, what they guarantee is an algorithm that will deliver the correct solution in *finite* time. Should this satisfy us? Finite time can be very long, and it is hard to distinguish a billion years from infinity. This is not just an abstract question for the working mathematician, as we recounted in Section 1.1. Indeed, both Haken's and Tarski's original algorithms were extremely slow, and computing their answer for objects of moderate size may indeed have required a billion years. Can this be quantified?

Efficient algorithms and computational complexity   This viewpoint suggests developing and using a computational yardstick and measuring the quality of understanding by the quality of the algorithms providing it. Indeed, we argue that better mathematical understanding of given mathematical structures often goes hand in hand with better algorithms for classifying their properties. Formalizing the notions of algorithms' resources (especially *time*) and their efficient use is the business of computational complexity theory, the subject of this book, which we shall start developing in the next section. But before we do, I would like to use the set of problems above to highlight a few other issues, which we will not discuss further here.

First, the *representation* of objects which algorithms process becomes important! One basic issue raised by most of the problems above is the contrast between continuity in mathematics and discreteness of computation. Algorithms manipulate finite objects (like bits) in discrete time steps. Knots, manifolds, dynamical systems, and the like are continuous objects. How can they be described to an algorithm and processed by it? As many readers will know, the answers vary, but finite descriptions exist for all. For example, we use knot diagrams for knots, triangulations for manifolds, and symbolic descriptions or successive approximations for dynamical systems. It is these *discrete* representations that are indeed used in algorithms for these (and other) *continuous* problems (as, e.g., Haken's algorithm demonstrates). Observe that this has to be the case; every continuous object we humans will ever consider has discrete representations! After all, math textbooks and papers are finite sequences of characters from a finite alphabet, just like the input to Turing machines. And we, their readers, would never be able to process and discuss them otherwise. All

this does not belittle the difficulties that may arise when seeking representations of continuous mathematical structures that would be useful for description, processing, and discussion—instead, it further illustrates the inevitable ties between mathematics and computation.

Let me demonstrate that algorithmic efficiency may crucially depend on representation even for simple *discrete* structures. Where would mathematics (and society) be if we continued using *unary* encodings of integers, or even Roman numerals? The *great* invention of the decimal encoding (or more generally, the *positional* number system) was motivated by, and came equipped with, efficient algorithms for arithmetic manipulation! And this is just one extremely basic example.

Problem 3 on the 4-colorability of planar maps points to a different aspect of the interaction of computation and mathematics. Many readers will know that problem (3) has a very simple decision procedure: Answer "yes" on every input. The correctness of this trivial algorithm is guaranteed by the highly nontrivial proof of the 4-color theorem of Appel and Haken [AH89]. This theorem states that in every planar map (as the ones used in geography texts, and Figure 2), each region can be colored from a set of 4 colors (e.g., Red, Blue, Green, Yellow), so that no two regions sharing a border are assigned the same color. This mathematical proof was the first to use a computer program as an essential tool to check that an enormously large but finite number of finite graphs is indeed 4-colorable. The proof naturally raised numerous discussions and arguments on the value of such proofs; discussions which only intensified with time, as more and more proofs use computers in a similar way (another famous example is Hales's proof of the Kepler conjecture [Hal05]). I leave it to the reader to contemplate the relative merits of computer vs. human generated proofs (and the task of distinguishing the two). Another point to make is that problem (3) may seem to some very different from the rest; all are well-known "deep" problems of mathematics, whereas (3) seems recreational. Indeed, in the twentieth century, it was quite popular in mathematics to call such problems "trivial" even without knowing the 4-color theorem, simply because a trivial brute-force algorithm that tries all (finitely many) possible colorings could determine the answer. This again brings us right back to the quality of algorithms, discussed next. You may want to revisit the task of distinguishing "deep" and "trivial" problems after reading Section 3.1.

3   Computational complexity 101: The basics, $\mathcal{P}$, and $\mathcal{NP}$

In this chapter, we develop the basic notions of computational problems; data representation; efficient computations; efficient reductions between problems; efficient verification of proofs; the classes $\mathcal{P}$, $\mathcal{NP}$, and co$\mathcal{NP}$; and the notion of $\mathcal{NP}$-complete problems. We focus on *classification* (or decision) problems; other types of problems, and classes, (enumeration, approximation, construction, etc.) are studied as well, and some will be discussed later in the book.

When studying efficiency, we will focus on *time* as the primary resource of algorithms. By "time", I mean the number of elementary operations[1] performed. Other resources of algorithms, such as memory, parallelism, communication, and randomness, are studied in computational complexity, and some will be treated later in the book.

3.1   Motivating examples

Let us consider the following three classification problems. As in Chapter 2, for each classification (or decision) problem like these, we get a description of an object and have to decide whether it has the desired property.

(1'.) Which Diophantine equations of the form $Ax^2 + By + C = 0$ are solvable by positive integers?

(2'.) Which knots on 3-dimensional manifolds bound a surface of genus $\leq g$?

(3'.) Which planar maps are 3-colorable?

Problem (1') is a restriction of problem (1) in Chapter 2. Problem (1) was undecidable, and it is natural to try to better understand more restricted classes of Diophantine equations. Problem (2') is a generalization of problem (2) in Chapter 2 in two ways: The unknotting problem (2) considers the special case of the manifold $\mathbb{R}^3$ and genus $g = 0$. Problem (2) was decidable, and so we may want to know whether its generalization (2') is decidable as well. Problem (3') is an interesting variant of problem (3). While every map has a 4-coloring, not every map has a 3-coloring; some do and some don't, and so this is another nontrivial classification problem to understand.

Most mathematicians would tend to agree that these three problems have absolutely nothing to do with one another. They are each from very different fields—algebra, topology, and combinatorics, respectively—each with its completely different notions, goals, and tools. However, the theorem below suggests that this view may be wrong.

Theorem 3.1. *Problems* (1') *,* (2') *and* (3') *are* equivalent.

Moreover, the equivalence notion is natural and completely formal. Intuitively, any understanding we have of one problem can be *simply* translated into a similar understanding of the other. The formal meaning of this equivalence will unfold in this chapter and be formalized in Section 3.9. To get there, we need to develop the language and machinery that yield such surprising results.

Representation issues   We start by discussing (informally and by example) how such varied complex mathematical objects can be described in finite terms, eventually as a sequence of bits. Often there are several alternative representations, and typically it is simple to convert one to the other. Let us discuss the representation of inputs in these three problems.

For problem (1') consider first the set of all equations of the form $Ax^2 + By + C = 0$ with integer coefficients A, B, C. A finite representation of such equations is obvious—the triple of coefficients (A, B, C), say with each integer written in binary notation. Given such a triple, the decision problem is whether the corresponding polynomial has a positive integer root (x, y). Let $2DIO$ denote the subset of triples for which the answer is YES.

---

[1] For example, reading/writing a finite amount of data from/to memory, or performing a logical or arithmetic operation involving a finite amount of data.

Finite representation of inputs to problem (2′) is tricky but still natural. The inputs consist of a 3-dimensional manifold M, a knot K embedded in it, and an integer G. A finite representation can describe M by a triangulation (a finite collection of tetrahedra and their adjacencies). The knot K will be described as a closed path along edges of the given tetrahedra. Given a triple (M, K, G), the decision problem is whether a surface that K bounds has genus at most G. Let *KNOT* denote the subset for which the answer is YES.

Finite representation of inputs to problem (3′) is nontrivial as well. Let us discuss not maps but instead graphs, in which vertices represent the countries and edges represent adjacency of countries (this view is equivalent; for a planar map, its graph is simply its dual map). To describe a graph (in a way that makes its planarity evident), one elegant possibility is to use a simple and beautiful theorem of Fáry [Fár48] (discovered independently by others and which has many proofs). It states that every planar graph has a *straight line* embedding in the plane (with no edges crossing). So, the input can be a set V of coordinates of the vertices (which can in fact be small integers) and a set E of edges, each a pair of elements from V. Let *3COL* be the subset of those inputs (V, E) describing a 3-colorable map.

Any finite object (integers, tuples of integers, finite graphs, finite complexes, etc.) can be represented naturally by binary sequences over the alphabet $\{0, 1\}$, and this is how they will be described as inputs to algorithms. As discussed above, even continuous objects like knots have finite descriptions and so can be described this way as well.[2] We will not discuss here subtle issues like whether objects have unique representations or whether every binary sequence should represent a legal object. It suffices to say that in most natural problems, this encoding of inputs can be chosen such that these are not real issues. Moreover, going back and forth between the object and its binary representation is simple and efficient (a notion to be formally defined below).

Consequently, let **I** denote the set of all finite binary sequences, and regard it as the set of inputs to all our classification problems. Indeed, every subset of **I** defines a classification problem. In this language, given a binary sequence $x \in \mathbf{I}$, we may interpret it as a triple of integers (A, B, C) and ask whether the related equation is in *2DIO*. This is problem (1′). We can also interpret x as a triple (M, K, G) of manifold, knot, and integer, and ask whether it is in the set *KNOT*. This is problem (2′), and the same can be done with (3′).

Reductions Theorem 3.1 states that there are *simple* translations (in both directions) between solving problem (1′) and problem (2′). More precisely, it provides efficiently computable functions $f, h \colon \mathbf{I} \to \mathbf{I}$ performing these translations:

$$(A, B, C) \in 2DIO \text{ iff } f(A, B, C) \in KNOT,$$

and

$$(M, K, G) \in KNOT \text{ iff } h(M, K, G) \in 2DIO.$$

Thus, an efficient algorithm to solve one of these problems immediately implies a similar one for the other. Putting it more dramatically, if we have gained enough understanding of topology to solve, say, the knot genus problem, it means that we automatically have gained enough number theoretic understanding for solving these quadratic Diophantine problems (and vice versa).

The translating functions f and h are called *reductions*. We capture the *simplicity* of a reduction in *computational* terms, demanding that it will be *efficiently* computable.

Similar pairs of reductions exist between the map 3-coloring problem and each of the other two problems. If sufficient understanding of graph theory leads to an efficient algorithm to determine whether a given planar map is 3-colorable, similar algorithms follow for both *KNOT* and *2DIO*. And vice versa—understanding either of them will similarly resolve 3-coloring. Note that this positive interpretation of the equivalence paints all three problems as equally "accessible." But the flip side says that they are also equally intractable: If any one of them lacks such an efficient classification algorithm, so do the other two. Indeed, with the better

---

[2]Theories of algorithms which have continuous inputs (e.g., real or complex numbers) have been developed, for example, in [BCSS98, BC06], but will not be discussed here.

understanding of these equivalences today, it seems more likely that the second interpretation is right: These problems are all hard to understand.

When teaching this material in class, or in lectures to unsuspecting audiences, it is always fun to watch listeners' amazement at these remarkably strong and unexpected connections between such remote problems. I hope it has a similar impact on you. But now it is time to dispel the mystery and explain the source of these connections. Here we go.

3.2   Efficient computation and the class $\mathcal{P}$

Efficient algorithms are the engine which drives an ever-growing part of industry and economy, and with it your everyday life. These jewels are embedded in most devices and applications you use daily. In this section we abstract a mathematical notion of efficient computation, polynomial-time algorithms. We motivate it and give examples of such algorithms.

In all that follows, we focus on asymptotic complexity. Thus, e.g., we care neither about the time it takes to factor the number $2^{67} - 1$ (as much as Mersenne cared about it), nor about the time it takes to factor all 67-bit numbers, but rather about the asymptotic behavior of factoring n-bit numbers, as a function of the input length n. The asymptotic viewpoint is inherent to computational complexity theory, and we shall see in this book that it reveals structure which would be obscured by finite, precise analysis. We note that the dependence on input size does not exist in Computability theory, where algorithms are simply required to halt in finite time. However, much of the methodology of these fields was imported to computational complexity theory—complexity classes of problems, reductions between problems and complete problems, all of which we shall meet.

Efficient computation (for a given problem) will be taken to be one whose runtime on any input of length n is bounded by a *polynomial* function in n.

Recall that $\mathbf{I}$ denotes the set of all binary sequences of all lengths. Let $\mathbf{I}_n$ denote all binary sequences in $\mathbf{I}$ of length n, namely $\mathbf{I}_n = \{0,1\}^n$.

Definition 3.2 (The class $\mathcal{P}$). A function $f \colon \mathbf{I} \to \mathbf{I}$ is in the class $\mathcal{P}$ if there is an algorithm computing f and positive constants A, c, such that for every n and every $x \in \mathbf{I}_n$, the algorithm computes $f(x)$ in at most An^c steps (namely, elementary operations).

Note that the definition applies in particular to Boolean functions (whose output is $\{0,1\}$), which capture classification problems (often called "decision problems"). We will abuse notation and sometimes think of $\mathcal{P}$ as the class containing *only* these classification problems. Observe that a function with a long output can be viewed as a sequence of Boolean functions, one for each output bit.

This important definition, contrasting polynomial growth with (brute force) exponential growth, was put forth in the late 1960s by Cobham [Cob65], Edmonds [Edm65b, Edm66, Edm67a], and Rabin [Rab67]. These researchers, coming from different areas and motivations, all attempted to formally delineate *efficient* from just *finite* algorithms. Edmonds's papers in particular supply some ingenious polynomial time algorithms to natural optimization problems. Of course, nontrivial polynomial-time algorithms were discovered earlier, long before the computer age. Many were discovered by mathematicians who needed efficient methods to calculate (by hand). The most ancient and famous example is of course Euclid's GCD (greatest common divisor) algorithm mentioned in Chapter 1, which was invented to bypass the need to factor integers when computing their largest common factor.

Two major choices must be made in selecting $\mathcal{P}$ to model the class of efficiently computable functions, which are often debated and certainly demand explanation. One is the choice of *polynomial* as the bound on time in terms of input length, and the second is the choice of *worst-case* requirement (namely, that this time bound holds for all inputs). We discuss the motivation and importance of these two choices below. However, it is important to stress that these choice are not dogmatic: computational complexity theory has considered and investigated numerous other alternatives to these choices. These include many finer grained bounds on efficiency other than polynomial, and many different notions of average case and input-dependent measures replacing the worst-case demands. Some of theses will be discussed later in the book. Still, the initial choices

above were extremely important in the early days of computational complexity, revealing beautiful structure that would become the solid foundation for the field, establish its methodology, and guide the subsequent study of finer and more diverse alternatives.

3.2.1 Why polynomial?

The choice of polynomial time to represent efficient computation seems arbitrary. However, this particular choice has justified itself over time from many points of view. I list some important justifications.

Polynomials typify "slowly growing" functions. The closure of polynomials under addition, multiplication, and composition preserves the notion of efficiency under natural programming practices, such as using two programs in sequence, or using one as a subroutine of another. This choice removes the necessity of describing the computational model precisely (e.g., it does not matter whether we allow arithmetic operations only on single digits or on arbitrary integers, since long addition, subtraction, multiplication, and division have simple polynomial-time algorithms taught in grade school). Similarly, we need not worry about data representation: one can efficiently translate between essentially any two natural representations of a set of finite objects.

From a practical viewpoint, a running time of, say, n^2 is far more desirable than n^{100}, and of course linear time is even better. Indeed even the constant coefficient of the polynomial running time can be crucial for real-life feasibility of an algorithm. However, there seems to be a "law of small numbers" at work: Very few known polynomial-time algorithms for natural problems have exponents above 3 or 4 (even though at discovery, the initial exponent may have been 30 or 40). However, many important natural problems that so far resist any efficient algorithms cannot at present be solved faster than in *exponential* time (which of course is totally impractical, even for small input data). This exponential gap gives great motivation for the definition of $\mathcal{P}$; reducing the complexity of such problems from exponential to (any) polynomial will be a huge conceptual improvement, likely involving new techniques.

3.2.2 Why worst case?

Another criticism of the definition of the class $\mathcal{P}$ is that a problem is deemed efficiently solvable if *every* input of length n can be solved in poly(n)-time. From a practical standpoint, it would suffice that the instances we care about (e.g., those generated by our application, be it in industry or nature) be solved quickly by our algorithms, and the rest can take a long time. Perhaps it suffices that "typical" instances be solved quickly. Of course, understanding what instances arise in practice is a great problem in itself, and a variety of models of typical behavior and algorithms for them are studied (and we shall touch upon this in section 4.4). The clear advantage of worst-case analysis is that we don't have to worry about which instances arise—they will all be solved quickly by what we call an "efficient algorithm". This notion "composes" well (i.e., when one algorithm is using another). Moreover, it accounts for adversarial situations where an input (or more generally, external behavior) is generated by an unknown opponent, who wishes to slow down the algorithm (or system). Modeling such adversaries is crucial in such fields as cryptography and error correction, and it is facilitated by worst-case analysis. Finally, as mentioned, this notion turned out to reveal a very elegant structure of the complexity universe, which inspired the more refined study of average-case and instance-specific theories.

Understanding the class $\mathcal{P}$ is central. Numerous computational problems arise (in theory and practice) that demand efficient solutions. Many algorithmic techniques were developed in the past four decades and enable solving many of these problems (see, e.g., the textbooks [CLR01, KT06]). These techniques drive the ultrafast home computer applications we now take for granted, like web searching, spell checking, data processing, computer game graphics, and fast arithmetic, as well as heavier duty programs used across industry, business, math, and science. But many more problems (some of which we shall meet soon), perhaps of higher practical and theoretical value, remain elusive. The challenge of *characterizing* this fundamental mathematical object—the class $\mathcal{P}$ of efficiently solvable problems—is far beyond us at this point.

We end this section with some examples of nontrivial problems in $\mathcal{P}$ of mathematical significance from diverse areas. In each, the interplay between mathematical and computational understanding needed for
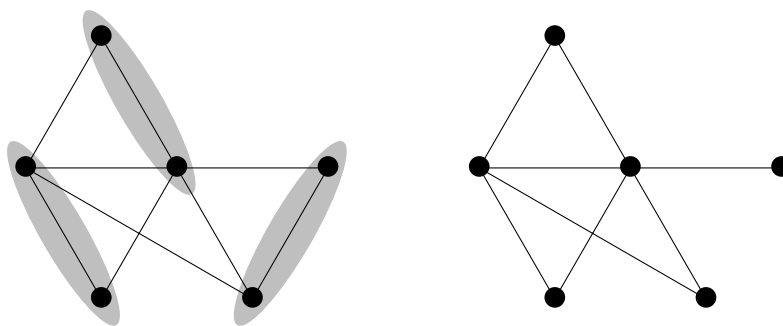
Figure 3. A graph with a perfect matching (left; matching is shown) and one without a perfect matching (right).

the development of these algorithms is evident. Most examples are elementary in nature, but if some mathematical notion is unfamiliar, feel free to ignore that example (or possibly better, look up its meaning).

3.2.3 Some problems in $\mathcal{P}$

- **Perfect matching.** Given a graph, test whether it has a *perfect matching*, namely, a pairing of all its vertices such that every pair is an edge of the graph (see Figure 3). The ingenious algorithm of Edmonds [Edm65b] is probably the first nontrivial algorithm in $\mathcal{P}$, and as mentioned above, this paper was central to highlighting $\mathcal{P}$ as an important class to study. The structure of matchings in graphs is one of the most well-studied subjects in combinatorics (see, e.g., [LP09]).

- **Primality testing.** Given an integer, determine whether it is prime.[3] Gauss literally appealed to the mathematical community to find an efficient algorithm, but it took two centuries to resolve. The story of this recent achievement of Agrawal, Kayal, and Saxena [AKS04] and its history is beautifully recounted by Granville in [Gra05]. Of course, there is no need to elaborate on how central prime numbers are in mathematics (and even in popular culture).

- **Planarity testing.** Given a graph, determine whether it is *planar*. Namely, can it be embedded in the plane without any edges crossing? (Try to determine this for the graphs in Figure 3 and those in Figure 5.) A sequence of *linear* time algorithms for this basic problem was discovered, starting with the paper of Hopcroft and Tarjan [HT74].

- **Linear programming.** Given a set of linear inequalities in many variables, determine whether they are mutually consistent, namely, whether there are real values for the variables satisfying all inequalities (a simple example is shown in Figure 4). This problem, and its optimization version, is enormously useful in applications. It captures many other problems (e.g., finding optimal strategies of zero-sum games). The convex optimization techniques used to give the efficient algorithms ([Kha79], [Kar84]) for it actually do much more (see, e.g., Schrijver's book [Sch03]).

- **Factoring polynomials.** Given a multivariate polynomial with rational coefficients, find its irreducible factors over $\mathbb{Q}$. The tools developed by Lenstra, Lenstra, and Lovász in [LLL82] (mainly regarding short bases in lattices in $\mathbb{R}^n$) have many other applications (see Section 13.8).

- **Hereditary graph properties.** Given a finite graph, test whether it is a member of some fixed minor-closed family.[4] A polynomial-time algorithm (which has a huge exponent in general) follows

---

[3]For example, try to determine the answer for $X - 1$ and $X + 1$, where $X = 6797727 \times 2^{15328}$.

[4]Namely, removing a vertex (or edge), and contracting an edge leave a graph in the family.

(a) A *feasible* linear program

(b) An *infeasible* linear program
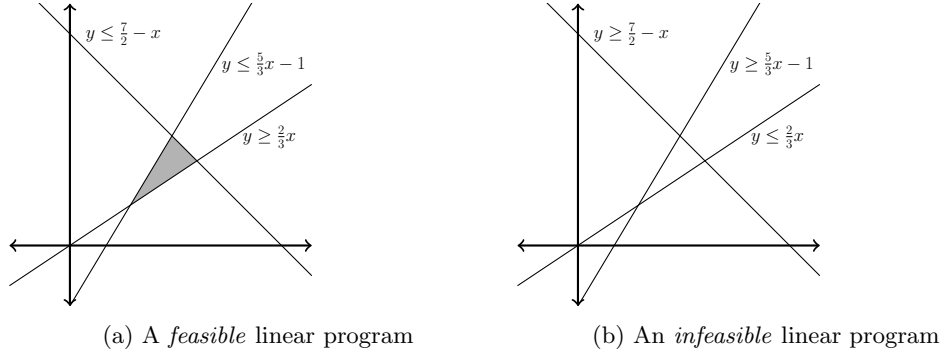
Figure 4. Two linear programs with two variables and three inequalities.

Robertson and Seymour's monumental structure theory [RS95] of such families, including a finite basis theorem.[5]

- **Permutation group membership.** Given a list of permutations on n elements, can the first one be generated by the rest?[6] The non-commutative Gaussian elimination techniques developed in [Sim70, FHL80] triggered the development of algorithmic group theory, which is used extensively by group theorists and in particular led to the breakthrough in testing graph isomorphism [Bab15].[7]

- **Hyperbolic word problem.** Given any presentation of a *hyperbolic* group by generators and relations, and a word w in the generators, check whether w represents the identity element. Gromov's geometric techniques, including isoperimetric bounds on the Cayley graphs of such groups [Gro87], allow a polynomial-time algorithm (and more). For general finitely presented groups, this problem is undecidable.

3.3   Efficient verification and the class $\mathcal{NP}$

Let $C \subset \mathbf{I}$ be a classification problem.[8] Specifically, we are given as input a binary sequence $x \in \mathbf{I}$ (describing a mathematical object), and are supposed to determine whether $x \in C$. It is convenient to view C as defining a *property*; $x \in C$ are objects having the property, and $x \notin C$ are objects that do not. If we have an efficient algorithm for C, we can simply apply it to x and know whether x has property C. But if we do not have such an algorithm, what is the next best thing? One answer is, a *convincing proof* that $x \in C$. Before defining it formally, let us see a couple of motivating examples.

The first example is a famous anecdote of a lecture given by F. N. Cole, titled "On the Factorization of Large Numbers," given at the 1903 American Math Society meeting. Without uttering a word, he went to the blackboard, wrote

$$2^{67} - 1 = 147573952589676412927 = 193707721 \times 761838257287,$$

and proceeded to perform the long multiplication of the integers on the right-hand side to derive the integer on the left: Mersenne's 67th number (which was conjectured to be prime). No one in the audience had any questions.

What happened there? Cole demonstrated that the number $2^{67} - 1$ is *composite*. Indeed, we can see that such a short proof can be given for any (correct) claim of the form $x \in COMPOSITES$, with $COMPOSITES$ denoting the set of composite numbers. The proof can simply be a factorization of x. The features we want

---

[5]Every such family has a finite number of excluded minors.

[6]A famous special case is the question, given a color pattern of a Rubik's cube (perhaps obtained by placing colored stickers illegally), can it be sorted to monochromatic faces by legal Rubik moves?

[7]See also the exposition [HBD17] of this result.

[8]In the computer science literature, C is often called a *language*.

to extract from this episode are two: the proofs here are *short* and *easily verifiable*. Indeed, the total length of the factors is roughly the length of the input, and multiplying them has an efficient algorithm. Note that the fact that it was extremely difficult for Cole to *find* these factors (he said it took him "three years of Sundays") did not affect in any way that demonstration.

A second example, which mathematicians meet daily, is what happens when we read a typical math journal paper. In it, we typically find a (claimed) theorem, followed by an (alleged) proof. Thus, we are verifying claims of the type $x \in THEOREMS$, where $THEOREMS$ is the set of all provable statements in, say, set theory. It is taken for granted that the written proof is *short* (page limit) and *easily verifiable* (a referee can verify it in a reasonable time), so at least on an intuitive level $THEOREMS$ has the same properties as $COMPOSITES$, and this can be made formal. Note again that we don't care how long it took the authors to *find* the proof. Needless to say, theorems and proofs in mathematical journals are not really written in a formal language; indeed, one can interpret the task of refereeing as verifying that these "semi-formal" proofs could be converted into formal ones that will establish the truth of the statements they claim to prove.

Now we are ready for a definition of $\mathcal{NP}$, the class of problems generalizing these two examples.

The class $\mathcal{NP}$ contains all properties C for which membership (namely, statements of the form $x \in C$) have *short, efficiently verifiable* proofs. As before, we use polynomials to define both terms. A candidate proof y for the claim $x \in C$ must have length at most polynomial in the length of x. And the verification that a given y indeed proves the claim $x \in C$ must be checkable in polynomial time (via a verification algorithm we will call V_C). Finally, if $x \notin C$, no such y should exist. Let us formalize this definition.

Definition 3.3 (The class $\mathcal{NP}$)**.** The set C is in the class $\mathcal{NP}$ if there is a function $V_C \in \mathcal{P}$ and a constant k such that

- If $x \in C$, then $\exists y$ with $|y| \leq k \cdot |x|^k$ and $V_C(x, y) = 1$;

- If $x \notin C$, then $\forall y$ we have $V_C(x, y) = 0$.

From a logic standpoint, each set C in $\mathcal{NP}$ may be viewed as a set of theorems in the complete and sound proof system defined by the *verification process* V_C.

A sequence y that "convinces" V_C that $x \in C$ is often called a *witness* or *certificate* for the membership of x in C. Again, we stress that the definition of $\mathcal{NP}$ is not concerned with how difficult it is to come up with a witness y, but rather only with the efficient verification using y that $x \in C$. The witness y (if it exists) can be viewed as given by an omnipotent entity, or simply guessed. Indeed, the acronym $\mathcal{NP}$ stands for "Nondeterministic Polynomial time," where the nondeterminism captures the ability of a *hypothetical* nondeterministic machine to "guess" a witness y (if one exists) and then verify it deterministically.

Nonetheless, the complexity of finding a witness is, of course, important, as it captures the *search problem* associated with $\mathcal{NP}$ sets. Every decision problem C (indeed, every verifier V_C for C) in $\mathcal{NP}$ defines a natural search problem associated with it: Given $x \in C$, *find* a short witness y that "convinces" V_C of this fact. A correct solution to this search problem can be efficiently verified by V_C, by definition.

It is clear that finding a witness (if one exists) can be done by brute-force search: as witnesses are short (of length poly(n) for a length-n input), one can enumerate all possible ones, and to each apply the verification procedure. However, this enumeration takes *exponential time* in n. The major question of this chapter (and this book, and the theory of computation!) is whether much faster algorithms than brute-force exist for *all* $\mathcal{NP}$ problems.

While it is usually the search problems that arise more naturally, it is often more convenient to study the decision versions of these problems (namely, whether a short witness exists). In almost all cases, both decision and search versions are computationally equivalent.[9]

Here is a list of some problems (or rather properties) in $\mathcal{NP}$, besides $THEOREMS$ and $COMPOSITES$ which we saw above. Note that some are variants of the problems in the similar list we gave for the class

---

[9]A notable possible exception is the set $COMPOSITES$ and the suggested verification procedure for it, accepting as witness a nontrivial factor. Note that while $COMPOSITES \in \mathcal{P}$ is a decision problem, the related search problem is equivalent to integer factorization, which is not known to have an efficient algorithm.

$\mathcal{P}$. However, we have no idea whether any of these are in $\mathcal{P}$. It is a good exercise (easy for most but not all examples) for the reader to define for each of them the short, easily verifiable witnesses for inputs having the property.[10]

Some problems in $\mathcal{NP}$:

- **Hamiltonian cycles in graphs.** The set of graphs having a Hamilton cycle, namely, a cycle of edges passing through every vertex exactly once (see Figure 5).[11]

- **Factoring integers.** Triples of integers (x, a, b), such that x has a prime factor in the interval $[a, b]$.

- **Integer programming.** Sets of linear inequalities in many variables that have an integer solution.

- **Matrix group membership.** Triples (A, B, C) of invertible matrices (say, over $\mathbb{F}_2$) of the same size, such that A is in the subgroup generated by B, C.

- **Graph isomorphism.** Pairs of graphs that are isomorphic, namely, having a bijection between their vertex sets that extends to a bijection on their edge sets. (Find which pairs of graphs in Figure 5 are isomorphic.)

- **Polynomial root**. Multivariate polynomials of degree 3 over $\mathbb{F}_2$ that have a root (namely, an assignment to the variables on which it evaluates to 0).
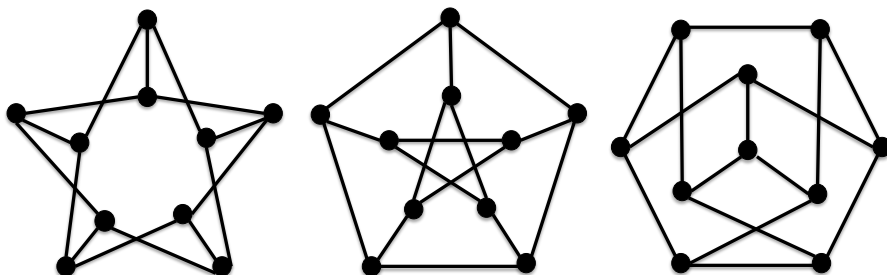
Figure 5. Which of these graphs are Hamiltonian? Which pairs of these graphs are isomorphic?

It is evident that decision problems in $\mathcal{P}$ are also in $\mathcal{NP}$. The verifier V_C is simply taken to be the efficient algorithm for C, and the witness y can be the empty sequence.

Corollary 3.4. $\mathcal{P} \subseteq \mathcal{NP}$.

But can we solve all $\mathcal{NP}$ problems efficiently? Can we vastly improve the trivial "brute-force" exponential time algorithm mentioned above to polynomial time for all $\mathcal{NP}$ problems? This is the celebrated $\mathcal{P}$ vs. $\mathcal{NP}$ question.

Open Problem 3.5. Is $\mathcal{P} = \mathcal{NP}$?

The definition of $\mathcal{NP}$, and the explicit $\mathcal{P} = \mathcal{NP}$? question (and much more that we will soon learn about) appeared formally first (independently and in slightly different forms) in the papers of Cook [Coo71] and Levin [Lev73] in the early 1970s, one researcher in America and the other in the Soviet Union. However, both the definition and question appeared informally earlier, again independently in the East and West, but with similar motivations. They all struggle with the *tractability* of solving problems for which *finite* algorithms

---

[10]The one difficult exception is Matrix Group Membership, which, if you cannot resolve yourself, peek in the beautiful [BS84].

[11]This problem is a special case of the well-known "Traveling Salesman Problem" (TSP), seeking the shortest such tour in a graph with edge lengths given.

exist, including finding finite proofs of theorems, short logical circuits for Boolean functions, isomorphism of graphs, and a variety of optimization problems of practical and theoretical interest. In all these examples, exhaustive search was an obvious but exponentially expensive solution, and the goal of improving it by a possibly more clever (and faster) algorithm was sought, hopefully one of polynomial complexity (namely, in $\mathcal{P}$). The key recognition was identifying the superclass $\mathcal{NP}$ that so neatly encompasses almost all the seemingly intractable problems mentioned above that people really cared about and struggled with.

The excellent survey of Sipser [Sip92] describes this history and gives excerpts from important original papers. Here I mention only a few precursors to the papers above. In the Soviet Union, Yablonskii and his school studied *Perebor*, literally meaning "exhaustive, brute-force search," and Levin's paper continues this line of research (see Trakhtenbrot's survey [Tra84] of this work, including a corrected translation of Levin's paper). In the West, Edmonds [Edm66] was the first to explicitly suggest "good characterization" of the short, efficiently verifiable type (which he motivates by a teacher-student interaction, although in a slightly stricter sense than $\mathcal{NP}$, which we will soon meet in Section 3.5). But already in 1956, a decade earlier, a remarkable letter (discovered only in the 1990s, see original and translation in [Sip92]) written by Gödel to von Neumann essentially introduces $\mathcal{P}$, $\mathcal{NP}$, and the $\mathcal{P}$ vs. $\mathcal{NP}$ question in rather modern language (see Section 1.2 of [Wig10b] for more). In particular, Gödel raises this fundamental problem of overcoming brute-force search, exemplifies that it is sometimes nontrivially possible, and demonstrates clearly how aware he was of the significance of this problem. Unfortunately, von Neumann was already dying of cancer at the time, and it is not known whether he ever responded or Gödel had further thoughts on the subject. It is interesting that these early papers have different expectations as to the resolution of the $\mathcal{P}$ vs. $\mathcal{NP}$ question (in the language of their time): Gödel speculates that *THEOREMS* might be in $\mathcal{P}$, while Edmonds [Edm67a] conjectures that the Traveling Salesman problem is not in $\mathcal{P}$.

A very appealing feature of the $\mathcal{P}$ vs. $\mathcal{NP}$ question (which was a source of early optimism about its possible quick resolution) is that it can be naturally viewed as a *bounded* analog of the decidability question from computability theory, which we already discussed implicitly in Chapter 2. To see this, replace the *polynomial-time* bound by a *finite* bound in both classes. For $\mathcal{P}$, the analog becomes all problems having finite algorithms, namely the decidable problems, sometimes called *Recursive* problems and denoted by $\mathcal{R}$. For $\mathcal{NP}$, the analog is the class of properties for which membership can be certified by a finite witness via a finite verification algorithm. This class of problems is called *Recursively Enumerable*, or $\mathcal{RE}$. It is easy to see that most problems mentioned in Chapter 1 are in this class. For example, consider the properties defined by problems 1 and 4 from Chapter 2, respectively the solvable Diophantine equations and the theorems provable in Peano arithmetic. In the first, an integer root of a polynomial is clearly a finite witness that can be easily verified by evaluation in finite time. In the second, a Peano proof of a given theorem is a finite witness, and the chain of deductions of the proof can be easily verified in finite time. Thus, both problems are in $\mathcal{RE}$. We already know that both problems are undecidable (namely, are not in $\mathcal{R}$) and so can conclude that $\mathcal{R} \neq \mathcal{RE}$.

With nearly a half century of experience, we realize that resolving $\mathcal{P}$ vs. $\mathcal{NP}$ is much harder than $\mathcal{R}$ vs. $\mathcal{RE}$. A possible analogy (with a much longer history) is the difficulty of resolving the Riemann Hypothesis, though we have known for millennia that there are infinitely many primes. In both contexts, what we already know is very qualitative, separating the finite and infinite, and what we want to know are very precise, quantitative versions. Also for both problems, some weak quantitative results were proved along the way. The prime number theorem is a much finer quantitative result about the distribution of primes than their infinitude. In this book, we will discuss analogous quantitative progress on the computational complexity of natural problems. In both cases, the long-term goals seem to require much deeper understanding of the respective fields and far better tools and techniques. Incidentally, we will discuss a completely different analogy between the $\mathcal{P}$ vs. $\mathcal{NP}$ question and the Riemann hypothesis in Chapter 8.

3.4   The $\mathcal{P}$ vs. $\mathcal{NP}$ question: Its meaning and importance

Should you care about the $\mathcal{P}$ vs. $\mathcal{NP}$ question? The previous sections make a clear case that it is a very important question of computer science. It is also a precise mathematical question. How about its importance for mathematics? For some mathematicians, the presence of this question in the list of seven

Clay Millennium Prize Problems [CJW06], alongside the Riemann Hypothesis and the Poincaré conjecture (which has since been resolved), may be sufficient reason to care. After all, these problems were selected by top mathematicians in the year 2000 as major challenges for the next millennium, each carrying a prize of one million dollars for its solution.

In this section, I hope to explain the ways in which the $\mathcal{P} = \mathcal{NP}$ question is unique not only among the Clay problems but also among all mathematics questions ever asked, in its immense practical and scientific importance, and its deep philosophical content. In a (very informal, sensational) nutshell, it can be summarized as follows:

Can we solve all the problems we can "legitimately" hope to solve?

where the royal "we" can stand for anyone or everyone, representing the general human quest for knowledge and understanding. In particular, this phrasing of the $\mathcal{P} = \mathcal{NP}$ question clearly addresses the possibility of resolving extant and future conjectures and open problems raised by mathematicians (at the very least, problems regarding classifications of mathematical objects).

To support this overarching interpretation of the $\mathcal{P} = \mathcal{NP}$ question, let us try to understand at a high level and in intuitive terms which problems occupy these two important classes. In fact, we have already intuitively identified the class $\mathcal{P}$ with a good approximation of all problems we can solve (efficiently, e.g., in our lifetimes). So next we embark on intuitively identifying $\mathcal{NP}$ as a good approximation of all "interesting" problems: those we are really investing effort in trying to solve, believing that we possibly can. Note that any argument for this interpretation will have to explain why undecidable problems (that are clearly not in $\mathcal{P}$) are not really "interesting" in this sense.

The very idea that all (or even most, or even very many) "interesting" problems can be mathematically identified is certainly audacious. Let us consider it, progressing slowly. We caution that this discussion is mainly of a philosophical nature, and the arguments I make here are imprecise and informal, representing my personal views. I encourage the reader to poke holes in these arguments, but I also challenge you to consider whether counterexamples found to general claims made here are typical or exceptional. After this section, we shall soon return to the sure footing of mathematics!

So, which problems occupy $\mathcal{NP}$? The class $\mathcal{NP}$ turns out to be extremely rich. There are literally thousands of $\mathcal{NP}$ problems in mathematics, optimization, artificial intelligence, biology, physics, economics, industry, and more that arise naturally out of very different necessities, and whose efficient solutions will benefit us in numerous ways. *What is common to all these possibly hard problems, which nevertheless separates them from certainly hard problems (like undecidable ones)?*

To explore this, it is worthwhile to consider a related question: *What explains the abundance of so many natural, important, diverse problems in the class $\mathcal{NP}$?* After all, this class was defined as a technical, mathematical notion by computational theorists. Probing the intuitive meaning of the definition of $\mathcal{NP}$, we will see that it captures many tasks of human endeavor *for which a successful completion can be easily recognized.* Consider the following professions, and the typical tasks they are facing (this list will be extremely superficial, but nevertheless instructive):

- **Mathematician:** Given a mathematical claim, come up with a proof for it.

- **Scientist:** Given a collection of data on some phenomena, find a theory explaining it.

- **Engineer:** Given a set of constraints (on cost, physical laws, etc.), come up with a design (of an engine, bridge, laptop, etc.) that meets these constraints.

- **Detective:** Given the crime scene, find "who done it."

Consider what may be a common feature of this multitude of tasks. I claim that in almost all cases, "we can tell" a good[12] solution when we see one (or we at least believe that we can). Simply put, *would you embark on a discovery process if you didn't expect to recognize what you set out to find?* It would be good for

---

[12]In this context, "good" may mean "optimal," or "better than previous ones," or "publishable," or any criterion we establish for ourselves.

the reader to consider this statement seriously and try to look for counterexamples. Of course, in different settings the "we" above may refer to members of the academic community, consumers of various products, or the juries in different trials. I have had many fun discussions, especially after lectures on the subject, of whether scientists or even artists are indeed in the mental state described. I believe they are. It seems to me that in these cases, the very decision to expose (or not) to others of our creations typically follows the application of such a "goodness test" to our work. Thus, embarking on any such task we undertake, we (implicitly or explicitly) expect the solution (or creation) we come up with to essentially bear the burden of proof of goodness that we can test; namely, be *short* and *efficiently verifiable*, just as in the definition of $\mathcal{NP}$.

The richness of $\mathcal{NP}$ follows from the simple fact that such tasks abound, and their mathematical formulation is indeed an $\mathcal{NP}$ problem. For all these tasks, *finding* solutions quickly is paramount, and so the importance of the $\mathcal{P}$ vs. $\mathcal{NP}$ question is evident. The colossal implications of the possibility that $\mathcal{P} = \mathcal{NP}$ are evident as well: as $\mathcal{P}$ represents efficiently solvable problems, we conclude that every instance of all these tasks can be solved efficiently.[13] Optimal solutions to humanity's most burning questions—medical, social, industrial, scientific, mathematical, and so forth—would be generated instantly (this is discussed in great detail and with many examples in Fortnow's popular book [For13]). A positive answer to one precise mathematical question holds the key to achieving this utopia! I believe that this universal promise seems to distinguish the $\mathcal{P}$ vs. $\mathcal{NP}$ question from every other mathematical question ever asked.

One can cast doubt on the strong statement above on several grounds. First, while most problems considered by humans may be of this nature (namely, their solutions are easily recognizable) using $\mathcal{P} = \mathcal{NP}$ (if true) to find these solutions requires the efficient recognition procedure to be fully specifiable formally. My reaction to this requirement is that for many important problems (especially in math, science, and engineering), such procedures already exist. For other important problems, if $\mathcal{P} = \mathcal{NP}$ is proved, there will be a huge incentive to convert intuitive recognition procedures into formal ones in order to use them. Another doubt may be that the polynomial-time algorithms supplied by $\mathcal{P} = \mathcal{NP}$ may be too slow in practice (e.g., because the polynomial time bound or the constants involved are too large). This indeed is possible and will in turn provide huge incentives to improve the efficiency of the algorithm. As discussed in Section 3.2, most of these problems currently have only brute-force, exponential-time algorithms, and a polynomial-time algorithm (even an inefficient one) must represent significant new understanding, which should now be fine-tuned to become more efficient.

So, should we believe that $\mathcal{P} = \mathcal{NP}$, and that such utopia is achievable? One (psychological) reason people feel that $\mathcal{P} = \mathcal{NP}$ is unlikely is that such tasks as listed above often require a degree of *creativity* or *ingenuity*, which people do not expect a simple computer program to have. We admire Wiles' proof of Fermat's last theorem; the scientific theories of Newton, Einstein, and Darwin; the design of the Golden Gate bridge and the pyramids; and sometimes even Hercule Poirot's and Miss Marple's analysis of a murder, precisely because they seem to require a leap that cannot be made by everyone, let alone by a simple mechanical device. I tend to disagree with this particular intuition. Note that these are all *specific* discoveries, namely, solving general problems on specific (highly important) instances. I see no reason that computers cannot do the same, as, after all, human brains (and all of nature) simply run efficient algorithms, like computers (nothing in our understanding of nature so far contradicts this observation, despite numerous speculations, writings, and beliefs to the contrary). So when we finally understand the algorithmic processes of the brain, we may indeed be able to automate the discovery of these specific achievements and perhaps many others. Indeed, the strides recently made on many frontiers of artificial intelligence suggest that computers will eventually outdo humans on almost every task.

But the question is whether we (humans or computers) can automate them *all*. Is it possible that for *every* task for which verification of solutions is easy, finding a solution is easy as well? If $\mathcal{P} = \mathcal{NP}$, the answer is positive, and creativity (of this universally abundant, verifiable kind) can be completely automated, on *every* instance. Most computer scientists (including myself) believe that this is not the case for the following, more mundane empirical reason. Attempts of probably millions of person-hours across industry and academia

---

[13]In fact, a much larger class of problems, which include some without any clear way of recognizing solutions, will also become easy to solve.

have been invested in proving that $\mathcal{P} = \mathcal{NP}$. This monumental effort was mostly inadvertent, made up of numerous independent projects (motivated mostly by potential profit of various applications, but also by mathematical curiosity) to find efficient algorithms for *specific* optimization problems. As will be explained in Section 3.8, if *any* of them had succeeded, a proof that $\mathcal{P} = \mathcal{NP}$ would follow. However, they all failed. Is this sufficient evidence? Hard to say, but here is the current widely held belief:

Conjecture 3.6. $\mathcal{P} \neq \mathcal{NP}$.

To segue into discussing the world of $\mathcal{P} \neq \mathcal{NP}$, I mention an important *negative* consequence of the $\mathcal{P} = \mathcal{NP}$ world, which perhaps makes it a bit less utopic than it seems. In this world, every code can be broken, practically disabling all Internet security and e-commerce applications as we know them. Indeed, it was the possibility that $\mathcal{P} \neq \mathcal{NP}$ that gave birth to *complexity-based* cryptography with its numerous applications used daily by all. Somehow, the existence of (specific, structured) hard problems whose solutions can be easily checked enables the creation of unbreakable codes between parties who have never before met (as is required, e.g., for online shopping) and many other seemingly impossible tasks.

It is quite striking that hard problems, which can't be solved, actually have applications! And so this world of $\mathcal{P} \neq \mathcal{NP}$, in which we probably live, has advantages as well. The nature of hardness required is discussed in Section 4.5, with its full utility exposited in the discussion of cryptography in Chapter 18. The intimate connection of hardness and randomness is discussed in Section 7.2.

Given this discussion, one may wonder why it is so hard to prove that indeed, $\mathcal{P} \neq \mathcal{NP}$—it may seem completely obvious that search is much harder than verification. We shall discuss attempts to do so and difficulties encountered in Chapter 5. Before that, in this chapter, we develop a methodology of reductions and completeness that enables us to identify the *hardest* problems in $\mathcal{NP}$, which might as well be the targets of any hardness proof. These developments and understanding, enlightening and important as they are, still seem to leave us far from the resolution of $\mathcal{P}$ vs. $\mathcal{NP}$.

While I have argued here for the huge importance of this question, its resolution (in either direction: $\mathcal{P} \neq \mathcal{NP}$ or $\mathcal{P} = \mathcal{NP}$) will only be the beginning, not the end of the story. As discussed, these classes are rather coarse, and are only two of many interesting ones. If we develop techniques to resolve $\mathcal{P}$ vs. $\mathcal{NP}$, one would hope they could be sharpened to determine far more precisely the computational resources needed to invest in solving specific problems!

Much discussion and other perspectives of the $\mathcal{P}$ vs. $\mathcal{NP}$ question, its meaning and importance, appear in all computational complexity texts and surveys referenced above, as well as in the newly published survey of Aaronson [Aar16b].

We will have much more to discuss about this question, but first we take a detour and turn to discuss a related question with a strong connection to mathematics: the $\mathcal{NP}$ vs. $\text{co}\mathcal{NP}$ question.

3.5   The class $\text{co}\mathcal{NP}$, the $\mathcal{NP}$ vs. $\text{co}\mathcal{NP}$ question, and efficiently characterizable structures

We have discussed efficient computation and efficient verification. Now let us turn to define and discuss efficient *characterization* of properties. Note that attempts, mainly in combinatorics, graph theory, and optimization, to find "good" characterizations (some successful ones, as for perfect matchings and Euler tours in graphs, and some failed ones, as for Hamiltonian cycles and colorings in graphs), were central to elucidating the definitions and importance of the concepts and classes in this chapter. Many of these, and the focus on formally defining the notion of "good" (in characterizations as well as in algorithms), go back Edmonds' early optimization papers, mainly [Edm66].

Fix a property $C \subseteq \mathbf{I}$. We already have the interpretations

- $C \in \mathcal{P}$ if it is easy to compute if an object x has property C,

- $C \in \mathcal{NP}$ if it is easy to certify that an object x has property C,

to which we now add

- $C \in \mathrm{co}\mathcal{NP}$ if it is easy to certify that an object x *does not have* property C,

where we formally define the class as follows.

Definition 3.7 (The class $\mathrm{co}\mathcal{NP}$)**.** A set C is in the class $\mathrm{co}\mathcal{NP}$ iff its complement $\bar{C} = \mathbf{I} \setminus C$ is in $\mathcal{NP}$.

For example, the set *PRIMES* of all prime numbers is in $\mathrm{co}\mathcal{NP}$, since its complement *COMPOSITES* is in $\mathcal{NP}$. Similarly, the set of non-Hamiltonian graphs is in $\mathrm{co}\mathcal{NP}$, since its complement, the set of all Hamiltonian graphs, is in $\mathcal{NP}$.

While the definition of the class $\mathcal{P}$ is symmetric,[14] the definition of the class $\mathcal{NP}$ is *asymmetric*. Having nice certificates that a given object has property C by no means automatically entails nice certificates that a given object does *not* have this property.

Indeed, when we can do both, namely, having nice certificates for both the set and its complement, we are achieving one of mathematics' holy grails of understanding structure, namely, *necessary and sufficient* conditions, sometimes phrased as a *characterization* or a *duality theorem*. As we well know, such characterizations are rare. When insisting (as I shall) that the certificates are furthermore *short, efficiently verifiable* ones,[15] such characterizations are even rarer. This leads to the following conjecture.

Conjecture 3.8. $\mathcal{NP} \neq \mathrm{co}\mathcal{NP}$.

Note that this conjecture implies $\mathcal{P} \neq \mathcal{NP}$. We shall discuss at length refinements of this conjecture in Chapter 6 on proof complexity.

Despite the shortage of such *efficient* characterizations (namely, properties that are simultaneously in $\mathcal{NP} \cap \mathrm{co}\mathcal{NP}$), they nontrivially exist. This class was introduced by Edmonds [Edm66], who called them problems with *good* characterization. Here is a list of some exemplary ones, following important theorems of (respectively) Menger, Dilworth, Farkas, von Neumann, and Pratt. I informally explain the $\mathcal{NP}$ and $\mathrm{co}\mathcal{NP}$ witnesses for most, which can be seen to be efficiently verifiable. Of course, the crux is that for each of these problems, *every* instance of the problem possesses one such witness: having the property or violating it.

Efficient duality theorems: problems in $\mathcal{NP} \cap \mathrm{co}\mathcal{NP}$

- **Graph k-connectivity.** The set of graphs in which *every* pair of vertices is connected by (a given number) k disjoint paths. Here the NP-witness is a collection of such k paths between every pair, and the $\mathrm{co}\mathcal{NP}$-witness is a *cut* of $k-1$ vertices whose removal disconnects some pair in the graph.

- **Partial order width.** Finite partially ordered set (poset) whose largest *antichain* (a set of pairwise incomparable elements) has at least (a given number) w elements. Here the $\mathcal{NP}$-witness is an antichain of w elements, and the $\mathrm{co}\mathcal{NP}$-witness is a partition of the given poset to $w-1$ *chains* (totally ordered sets).

- **Linear programming.** Systems of consistent linear inequalities. Here an $\mathcal{NP}$-witness is a point satisfying all inequalities. A $\mathrm{co}\mathcal{NP}$-witness is a linear combination of the inequalities producing the contradiction $0 > 1$.[16]

- **Zero-sum games.**[17] Finite zero-sum games (described by a real payoff matrix) in which the first player can gain at least v (some given value). Here the $\mathcal{NP}$-witness is a strategy for the first player (namely, a probability distribution on the rows), which guarantees her a payoff of v, and the $\mathrm{co}\mathcal{NP}$-witness is a strategy for the second player (namely, probability distribution on the columns), which guarantees that he pays less than v.

---

[14] Having a fast algorithm to determine whether an object has a property C is equivalent to having a fast algorithm for the complementary set $\bar{C}$. In other words, $\mathcal{P} = \mathrm{co}\mathcal{P}$.

[15] There are many famous duality theorems in mathematics that do not conform to this strict efficiency criterion (e.g., Hilbert's Nullstellensatz).

[16] This duality generalizes to other convex bodies given by more general constraints, like *semi-definite* programming. Such extensions include the Kuhn-Tucker conditions and the Hahn-Banach theorem.

[17] This problem was later discovered to be equivalent to linear programming.

- **Primes.** Prime numbers. Here the $\mathrm{co}\mathcal{NP}$-witness is simple: two nontrivial factors of the input. The reader is encouraged to attempt to find the $\mathcal{NP}$-witness: a short certificate of primality. It requires only very elementary number theory.[18]
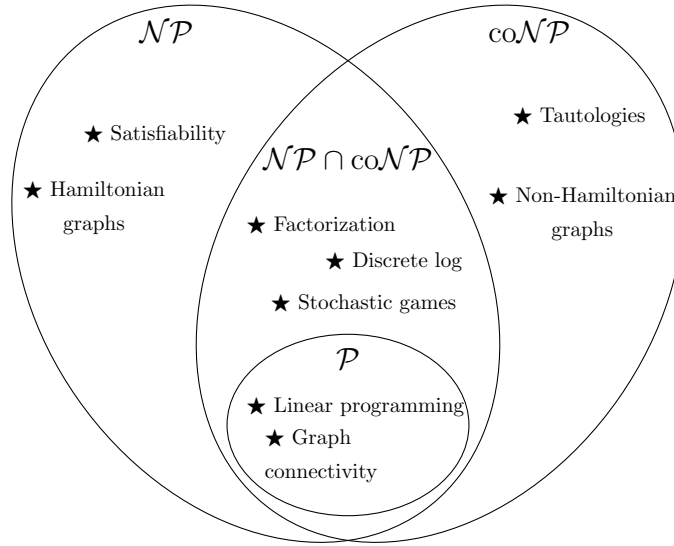
Figure 6. $\mathcal{P}$, $\mathcal{NP}$, and $\mathrm{co}\mathcal{NP}$.

The known relations of $\mathcal{P}$, $\mathcal{NP}$, and $\mathrm{co}\mathcal{NP}$ are depicted in Figure 6. The examples above, of problems in $\mathcal{NP} \cap \mathrm{co}\mathcal{NP}$ were chosen to make a point. At the time of their discovery, interest was seemingly focused only on characterizing these structures; it is not clear whether efficient algorithms for these problems were sought as well. However, with time, all these problems turned out to be in $\mathcal{P}$, and their resolutions entered the Hall of Fame of efficient algorithms. Famous examples are the ellipsoid method of Khachian [Kha79] and the interior-point method of Karmarkar [Kar84], both for Linear Programming, and the breakthrough algorithm of Agrawal, Kayal, and Saxena [AKS04] for Primes.[19]

Is there a moral to this story? Only that sometimes, when we have an efficient characterization of structure, we can hope for more: efficient algorithms. Indeed, a natural stepping stone toward an elusive efficient algorithm may first be to get an efficient characterization.

Can we expect this magic to always happen? Is $\mathcal{NP} \cap \mathrm{co}\mathcal{NP} = \mathcal{P}$? We do not have too many examples of problems in $\mathcal{NP} \cap \mathrm{co}\mathcal{NP}$ that have resisted efficient algorithms. Some of the famous, like integer factoring and discrete logarithms,[20] arise from *one-way* functions which underlie cryptography (we discuss these in Section 4.5). Note that while they are not known to be hard, humanity literally banks on their intractability for electronic commerce security. Yet another famous example, for which membership in $\mathcal{NP}$ and in $\mathrm{co}\mathcal{NP}$ are highly nontrivial (respectively proved in [Lac15] and [HLP99]) is the *unknottedness* problem, namely, testing whether a knot diagram represents the trivial knot. A very different example is Shapley's *stochastic games*, studied by Condon in [Con92], for which no efficient algorithm is known. However, we have seen that many problems first proved to be in $\mathcal{NP} \cap \mathrm{co}\mathcal{NP}$ eventually were found to be in $\mathcal{P}$. It is hard to generalize from so few examples, but the general belief is that the two classes are different.

---

[18]Hint: Roughly, the witness consists of a generator of Z_p^*, a factorization of $p-1$, and a recursive certificate of the same type for each of the factors.

[19]It is interesting that assuming the Generalized Riemann Hypothesis, a simple polynomial-time algorithm was given 30 years earlier by Miller [Mil76].

[20]Which have to be properly defined as decision problems.

Conjecture 3.9. $\mathcal{NP} \cap \text{co}\mathcal{NP} \neq \mathcal{P}$.

Note that this conjecture implies $\mathcal{P} \neq \mathcal{NP}$.

We now return to developing the main mechanism, which will help us study such questions: *efficient reductions* and *completeness*.

3.6   Reductions: A partial order of computational difficulty

In this section, we deal with relating the computational difficulty of problems for which we have no efficient solutions (yet).

Recall that we can regard any classification problem (on finitely described objects) as a subset of our set of inputs $\mathbf{I}$. Efficient reductions provide a natural partial order on such problems that captures their relative difficulty. Note that reductions are a primary tool in computability and recursion theory, from which computational complexity developed. There, reductions were typically simply *computable* functions, whereas the focus of computational complexity is on *efficiently computable* ones. While we concentrate here on time efficiency, the field studies a great variety of other resources; limiting these in reductions is as fruitful as limiting them in algorithms. The following crucial definition is depicted in Figure 7.

Definition 3.10 (Efficient reductions). Let $C, D \subset \mathbf{I}$ be two classification problems. $f \colon \mathbf{I} \to \mathbf{I}$ is an efficient reduction from C to D if $f \in \mathcal{P}$ and for every $x \in \mathbf{I}$ we have $x \in C$ iff $f(x) \in D$. In this case, we call f an *efficient reduction* from C to D. We write $C \leq D$ if there is an efficient reduction from C to D.
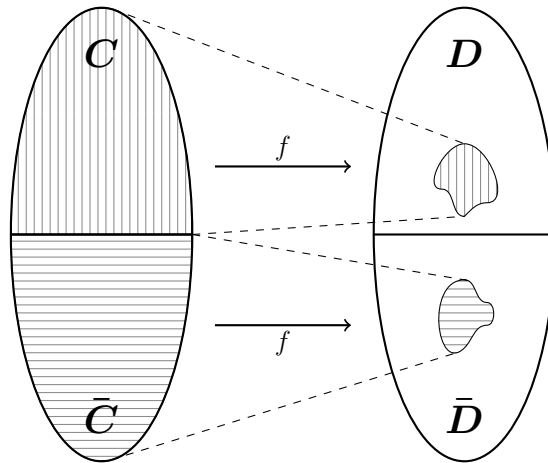
Figure 7. A schematic illustration of a reduction between two classification problems.

So, if $C \leq D$, then solving the classification problem C is computationally not much harder than solving D (up to polynomial factors in the running time).[21] If we have both $C \leq D$ and $D \leq C$ then C and D are computationally equivalent (again, up to polynomial factors). This gives formal meaning to the word "equivalent" in Theorem 3.1.

The definition of efficient computation (more precisely, that the composition of two polynomials is a polynomial) allows two immediate but important observations on the usefulness of efficient reductions. Please verify both for yourself! First, that indeed $\leq$ is *transitive*, and thus defines a partial order on classification problems. Second, one can compose (as in Figure 8) an efficient algorithm for one problem and an efficient reduction from a second problem to get an efficient algorithm for the second. Specifically, if $C \leq D$ and $D \in \mathcal{P}$, then also $C \in \mathcal{P}$.

---

[21]In particular, if $C \in \mathcal{P}$ then it is not much harder than trivial problems D (e.g. D might ask to distinguish sequences starting with 0 from those starting with 1). The reduction in this case would simply solve C.
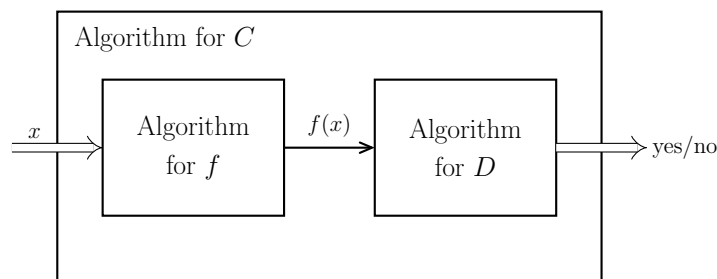
Figure 8. Composing a reduction and an algorithm to create a new algorithm.

As noted, $C \leq D$ means that solving the classification problem C is computationally not much harder than solving D. In some cases, one can replace "computationally" by the (vague) term "mathematically." To be useful mathematically and allow better understanding of one problem in terms of another may require more properties of the reduction f than merely being efficiently computable. For example, we may want f to be a linear transformation, or a low-degree polynomial map, and indeed in some cases (as we will see e.g. in Chapter 12) this is possible. When such a connection between two classification problems (which look unrelated) can be proved, it can mean the importability of techniques from one area to another.

The power of efficient reductions to relate seemingly unrelated notions will unfold in later sections. We shall see that they can relate not only classification problems but also such diverse concepts as hardness to randomness; average-case to worst-case difficulty; proof length to computation time; and last but not least, the security of electronic transactions to the difficulty of factoring integers. In a sense, *efficient reductions are the backbone of computational complexity*. Indeed, given that polynomial time reductions can do all these wonders, no wonder we have a hard time characterizing the class $\mathcal{P}$!

3.7   Completeness: Problems capturing complexity classes

We now return to classification problems. The partial order of their difficulty, provided by efficient reductions, allows us to define the *hardest* problems in a given class of problems. Let $\mathcal{C}$ be any collection of classification problems (namely, every element of $\mathcal{C}$ is a subset C of $\mathbf{I}$). In this chapter we are mainly concerned about the class $\mathcal{C} = \mathcal{NP}$. But later in the book we will see this important idea recur for other *complexity classes* (namely classes of problems defined by the resources required to solve them, like $\mathcal{NP}$) .

Definition 3.11 (Hardness and completeness)**.** A problem D is called $\mathcal{C}$-*hard* if for every $C \in \mathcal{C}$, we have $C \leq D$. If we further have that if $D \in \mathcal{C}$, then D is called $\mathcal{C}$-*complete*.

In other words, if D is $\mathcal{C}$-complete, it is a hardest problem in the class $\mathcal{C}$: if we manage to solve D efficiently, we have done so automatically for all other problems in $\mathcal{C}$.

It is not a priori clear that a given class has any complete problems! It is perhaps a miracle of computational complexity that many natural *complexity* classes do! Note that a given class may have many complete problems, and by definition, they all have essentially the same complexity. If we manage to prove that *any* of them cannot be efficiently solved, then we automatically have done so for *all* of them.

It is trivial (and uninteresting) that every problem in the class $\mathcal{P}$ is in fact $\mathcal{P}$-complete under our definition. It becomes interesting when we find such universal problems in classes of problems for which we do not have efficient algorithms. By far, the most important of all such classes is $\mathcal{NP}$.

3.8 $\mathcal{NP}$-completeness

As mentioned earlier, the seminal papers of Cook [Coo71] and Levin [Lev73] defined $\mathcal{NP}$, efficient reducibilities and completeness, but their crowning achievement was the discovery of a *natural* $\mathcal{NP}$-complete problem.

Definition 3.12 (The problem SAT). A Boolean formula is a logical expression over Boolean variables (that can take values in $\{0,1\}$) with connectives $\wedge, \vee, \neg$ (standing for AND, OR, NOT), for example, $(x_1 \vee x_2) \wedge (\neg x_3)$. Let SAT denote the set of all *satisfiable* Boolean formulas (namely, those for which a Boolean assignment to the variables evaluates to 1).

For example, the following formula is unsatisfiable:

$$(\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_3) \wedge (\neg x_3) \wedge (x_1 \vee x_3 \vee x_4) \wedge (\neg x_4 \vee x_1),$$

while the one below is satisfiable:

$$(\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_3) \wedge (x_1 \vee x_3 \vee x_4) \wedge (\neg x_4 \vee x_1).$$

We now arrive at a foundational theorem of computational complexity, revealing the primary importance of this simple-looking problem of satisfying Boolean formulas.

Theorem 3.13 [Coo71, Lev73]. *SAT is $\mathcal{NP}$-complete.*

Recall the meaning of that statement. For *every* set $C \in \mathcal{NP}$, there is an efficient reduction $f \colon \mathbf{I} \to \mathbf{I}$ with the following property: for every sequence $z \in \mathbf{I}$, we have that $z \in C$ iff the sequence $f(z)$ encodes a satisfiable formula.

The proof of this theorem (namely, the construction of the reduction algorithm f) gives an extra bonus that turns out to be extremely useful: It maps witnesses to witnesses.[22] More precisely, for any witness y certifying that $z \in C$ (via some verifier V_C), the same reduction f converts the witness y to a Boolean assignment of the variables of the formula $f(z)$ that satisfy it. In other words, this reduction translates not only between the *decision* problems, but also between the associated *search* problems.

A few words are in order about the proof of Theorem 3.13. It is of course easy to see that SAT is in $\mathcal{NP}$; a satisfying assignment is an easy-to-verify witness. The difficult part is proving the $\mathcal{NP}$-hardness of SAT. Certainly the proof cannot afford to consider every problem $C \in \mathcal{NP}$ separately. The gist of the proof is a *generic* transformation: taking a description of the verifier V_C for C and emulating its computation on input z and hypothetical witness y to efficiently create a Boolean formula $f(z)$ (whose variables are the bits of y). The formula constructed simply tests the validity of the computation of V_C on (z, y) and checks that this computation outputs 1. Here the locality and simplicity of the individual steps of algorithms (say, described as Turing machines) play a central role: Checking the consistency of each step of the computation of V_C amounts essentially to a constant size formula on a few bits.

To summarize, SAT captures the difficulty of the whole class $\mathcal{NP}$. In particular, the $\mathcal{P}$ vs. $\mathcal{NP}$ problem can now be phrased as a question about the complexity of *one* problem instead of infinitely many of them.

Corollary 3.14. $\mathcal{P} = \mathcal{NP}$ *iff* $SAT \in \mathcal{P}$.

A great advantage of having one complete problem at hand (like SAT) is that now, to prove that another problem (say, $D \in \mathcal{NP}$) is $\mathcal{NP}$-complete, we only need to design a reduction from SAT to D (namely, prove $SAT \leq D$). We already know that for *every* $C \in \mathcal{NP}$, we have $C \leq SAT$, and transitivity of $\leq$ takes care of the rest.

This idea was powerfully used in the next seminal paper, that of Karp [Kar72]. In his paper, Karp listed 21 problems from logic, graph theory, scheduling, and geometry, and showed them to be $\mathcal{NP}$-complete. This was the first demonstration of the wide spectrum of $\mathcal{NP}$-complete problems, and it initiated an industry of finding more such problems. A few years later, Garey and Johnson [GJ79] published their book on $\mathcal{NP}$-completeness, which contains hundreds of such problems from diverse branches of science, engineering, and mathematics. Today, thousands of these problems are known. We will soon discuss the meaning and importance of this notion, but first I give some examples of $\mathcal{NP}$-complete problems, and the nature of the connection among them.

---

[22]We will see one crucial use of this property in the proof of Theorem 10.5 on *zero-knowledge proofs*.

3.9   Some $\mathcal{NP}$-complete problems

We stress again that all $\mathcal{NP}$-complete problems are equivalent in a very strong sense. Any algorithm solving one can be simply translated into an equally efficient[23] algorithm solving any other.

We are finally ready to see the proof of Theorem 3.1 on the equivalence of our motivating examples from Section 3.1. It follows from the following three theorems.

Theorem 3.15 [AM75]. *The set 2DIO is $\mathcal{NP}$-complete.*

Theorem 3.16 [AHT06]. *The set KNOT is $\mathcal{NP}$-complete.*

Theorem 3.17 [Kar72, Sto73]. *The set 3COL is $\mathcal{NP}$-complete.*

Recall that to prove $\mathcal{NP}$-completeness of a set, one has to prove two things: that it is in $\mathcal{NP}$, and that it is $\mathcal{NP}$-hard. In almost all $\mathcal{NP}$-complete problems, membership in $\mathcal{NP}$ (namely, the existence of short certificates) is easy to prove. Certainly, a candidate 3-coloring of a given map is short and easy to check. For 2DIO, one can easily see that if there is a positive integer solution (x, y) to $Ax^2 + By + C = 0$, then in fact there is a short one,[24] indeed a solution whose length (in bits) is linear in the lengths of A, B, C. Thus, a short witness is simply a root (x, y). But KNOT is an exception, and the short witnesses for the knot having a small genus requires Haken's algorithmic theory of normal surfaces, considerably enhanced (even short certificates for unknottedness in $\mathbb{R}^3$ are hard to obtain; see [HLP99]). Let us discuss what these $\mathcal{NP}$-completeness results mean, first about the relationship between the three sets, and then about each individually.

The proofs that these problems are complete follow by reductions from (variants of) SAT. The discrete, combinatorial nature of these reductions may cast doubt on the possibility that the computational equivalence of these problems implies the ability of real "technology transfer" between, for example, topology and number theory. Nevertheless, now that we know of the equivalence, perhaps simpler and more direct reductions can be found between these problems. Moreover, we stress again that reductions translate between witnesses as well. Namely, for any instance, say $(M, K, G) \in KNOT$, if we translate it using this reduction to an instance $(A, B, C) \in 2DIO$ and happen (either by sheer luck or special structure of that equation) to find an integer root, the same reduction will translate that root back to a description of a genus G manifold that bounds the knot K. Today many such $\mathcal{NP}$-complete problems are known throughout mathematics, and for some pairs, the equivalence can be mathematically meaningful and useful (as it is between some pairs, of computational problems).

Let us discuss the simplest of the three reductions above, namely, from SAT to 3COL. If you have never seen one, it should be a mystery: The two problems talk about different worlds, one of logic and the other of graph theory. Both are difficult problems, but the reduction should be easy (namely, efficiently computable). The key to this reduction, as well as to almost any other, is the locality of computation! This of course is evident in SAT; a formula is composed from Boolean gates, each of which performs a simple, local operation. However, 3COL feels like a more global property.[25] The idea of this reduction is to focus on the individual gates of the input formula. We'll find a reduction that works for each gate and will compose the small ("gadget") graphs produced, mimicking the structure prescribed by the input formula. Let's elaborate this idea.

Here is how to transform the satisfiability problem for the (trivial, 1-gate) formula $x \lor y$ to a graph 3-coloring problem. We will actually transform the equation $x \lor y = z$ to a graph 3-coloring statement using the gadget graph shown in Figure 9. Check that it satisfies the following condition: In *every* legal 3-coloring of the graph with colors $\{0, 1, 2\}$, the colors of the vertices labeled x, y, z will be from $\{0, 1\}$, which will satisfy the equation $x \lor y = z$. One can easily construct such gadgets for the gates $\land$, $\neg$ as well. Now, to complete the reduction, the algorithm proceeds as follows. Given an arbitrary formula as input, it names its wires, builds a gadget graph for every gate, and identifies appropriate vertices in these to generate an output graph. By construction, it is 3-colorable if and only if the given formula was satisfiable. This is essentially

---

[23]As usual, up to polynomial factors.

[24]Hint: If (x, y) is a root, so is $(x + B, y - A(2x + B))$.

[25]Consider, for example, a cycle on n vertices, where n is odd; it requires 3 colors, but if we remove any edge, it can be 2-colored.

the reduction in [Kar72], but we are not done yet: The gadget graph above is not planar (and hence the output graphs are also not planar). However, Stockmeyer [Sto73] gives another gadget that can eliminate crossings in planar embeddings of graphs without changing their 3-colorability. The reader is encouraged to find such a gadget. With this, the proof of Theorem 3.17 is complete.
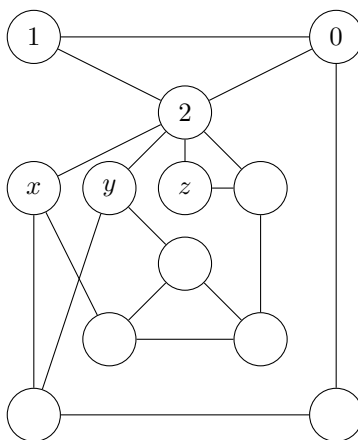
Figure 9. The gadget underlying the reduction from SAT to $3COL$.

We now list a few more $\mathcal{NP}$-complete problems of a different nature, to give a feeling for the breadth of this phenomenon. Some appear already in Karp's original article [Kar72]. Again, hundreds more can be found in Garey and Johnson's book, [GJ79], and by now, many thousands are known:

- **Hamiltonian cycle.** Given a graph, is there a simple cycle of edges going through every vertex precisely once?

- **Subset-sum.** Given a sequence of integers $a_1, \ldots, a_n$ and b, is there a subset J such that $\sum_{i \in J} a_i = b$?

- **Integer programming.** Given a polytope in $\mathbb{R}^n$ (by its bounding hyperplanes), does it contain an integer point?

- **Clique.** Given a graph and an integer k, are there k vertices with all pairs mutually adjacent?

- **Quadratic equations.** Given a system of multivariate polynomial equations of degree at most 2, over a finite field (say, $\mathbb{F}_2$), do they have a common root?

- **Shortest lattice vector.** Given a lattice L in $\mathbb{R}^n$ and an integer k, is the shortest nonzero vector of L of (Euclidean) length $\leq k$?

3.10   The nature and impact of $\mathcal{NP}$-completeness

$\mathcal{NP}$-completeness is a unique scientific discovery—there seems to be no precisely defined scientific notion that even comes close to being pervasive in so many fields of science and engineering! We start with its most immediate impact, in computer science itself, move to mathematics, and then to science and beyond. Some of this discussion will be become more meaningful (and impressive) as you read further through the book, and in detail in the last chapter. More can be found in, for example, Papadimitriou's retrospective on the subject [Pap97]. Curiously, that paper reports that electronic search (new at the time) revealed thousands of science and math papers with the phrase "$\mathcal{NP}$-complete" in them; today, more than 20 years later, this number is far larger!

As mentioned, starting with Karp's paper [Kar72], an explosion of $\mathcal{NP}$-completeness results followed quickly in every corner and subfield of computer science. This is easy to explain. Most individuals in the

field of computer science, from academics to industry programmers, are busy seeking efficient algorithms for numerous computational problems. How can one justify failure to find such an algorithm? In the absence of any techniques for proving intractability, the next best thing was proving that the computational problem at hand was $\mathcal{NP}$-complete (or $\mathcal{NP}$-hard), which means that finding such an efficient algorithm for it would imply an efficient algorithm for numerous others, which many others failed to solve. In short, failing to prove $\mathcal{P} = \mathcal{NP}$ is a very powerful excuse, and $\mathcal{NP}$-completeness is an excellent stamp of hardness. Every professional of the field knows this! While $\mathcal{NP}$-completeness is a negative result (basically showing that what we want is impossible), such negative results had a positive impact. As problems do not go away when you declare them $\mathcal{NP}$-complete and still demand solutions, weaker solution concepts for them were developed. For example, for optimization problems, people attempted to find good approximation algorithms. Moreover, given that $\mathcal{NP}$-completeness only captures worst-case hardness, people developed algorithms that work well "on average" and heuristics that seem to work well on inputs that "show up in practice." This direction needs lots more theory, especially in light of the recent practical success of *deep networks*, that is far from understood. A variety of quality criteria and models were developed for different relaxations of efficient solvability, leading to analogous complexity theories, which enable researchers to argue hardness as well; some of these models will be discussed later in the book. A major such theory, able to argue $\mathcal{NP}$-completeness for approximation problems, and actually pinpoint in many cases the exact limits of efficient approximation, will be discussed in Section 10.3.

The next field to be impacted by $\mathcal{NP}$-completeness was mathematics. With some delay, $\mathcal{NP}$-completeness theorems started showing up in most mathematical disciplines, including algebra, analysis, geometry, topology, combinatorics, and number theory. This "intrusion" may seem surprising, as most questions that mathematicians ask themselves are not algorithmic. However, existence theorems for a variety of objects beg the question of having "explicit" descriptions of such objects. Moreover, in many fields, one actually needs to find such objects. Mathematics is full of a variety of constructions, done by hand long ago and by numerous libraries of computer programs that are essential for progress, and hence, their efficiency is also essential. Like computer scientists, mathematicians adopted the notion of a polynomial-time algorithm as a first cut at defining "efficient" and "explicit." Thus, description and construction problems that are $\mathcal{NP}$-complete were extremely useful to set limits on the hopes of achieving these properties. Furthermore, in mathematics, such $\mathcal{NP}$-completeness results signified an underlying "mathematical nastiness" of the structures under study. For example, as explained in Section 3.5, an efficient characterization of a property that is $\mathcal{NP}$-complete will imply that $\mathcal{NP} = \text{co}\mathcal{NP}$, and so it is unlikely (as we understand things today) that such a characterization exists. As in CS, so in math as well, such bad news begets good outcomes, nudging mathematicians into more productive directions: refining or specializing the properties under study, considering a variety of approximate notions, or simply being satisfied with sufficient and necessary conditions that are not complementary (as is needed for characterization).

The presence and impact on $\mathcal{NP}$-completeness in science is evidenced by the fact that such results (which are patently about computation) in biology, chemistry, economics, neuroscience, electrical engineering, and other fields are being proved not by computer scientists but by biologists, chemists, economists, neuroscientists, electrical engineers, and so forth. Moreover, these results are being published in the scientific journals of these very fields. And the numbers are staggering: A search for papers that contain the phrase "$\mathcal{NP}$-complete" or "$\mathcal{NP}$-completeness" prominently (in the title, abstract, or keywords) reveals that in *each* of these disciplines, there are *hundreds* of such papers, and many thousands more that mention these terms in the body of the paper. To obtain such results, these thousands of scientists needed to learn the concepts and proof methods of computational complexity, typically a foreign language to most (for instance, mathematical theorems rarely appear in science articles at all).

This phenomenon begs an explanation! Indeed, there are two questions to answer. What explains the abundance of $\mathcal{NP}$-completeness in these diverse disciplines? And why do their scientists bother making the unusual effort to prove these computational theorems?

One important observation is that scientists often study processes and try to build models that explain and predict them. Almost by definition, these are computational processes, namely, composed of a sequence of *simple, local steps*, like Turing machines, albeit manipulating not bits in computers but possibly neurons

in the brain, proteins in the cell, atoms in matter, fish in a school, or stars in a galaxy. In other words, many models simply describe *algorithms* that nature uses for generating certain processes or behavior. A typical $\mathcal{NP}$-completeness result often refers to the limits of prediction by a particular model of some natural process. Here are some illustrative examples. In some existing models, it is $\mathcal{NP}$-complete to compute the following quantities: the minimal surface area that a given foam will settle into (in physics), the minimal energy configuration of a certain molecule (e.g., as in protein folding in biology), and the maximum social welfare of certain equilibria (in economics). Let's explore the meaning of such results to modeling natural phenomena.

We'll make two natural assumptions, which seem completely benign. First, that $\mathcal{P} \neq \mathcal{NP}$. Then $\mathcal{NP}$-completeness means that no efficient algorithm can compute the required quantities (e.g., in the examples just mentioned), at least for *some* instances. Second, that natural processes are inherently efficient algorithms, and so are the measurements we perform to extract these quantities at the end of the process. These two assumptions clash with each other, which seems to suggest at least one of two possible conclusions. One possibility is that the model (for which $\mathcal{NP}$-completeness was proved) is simply wrong (or incomplete) in describing reality. The other possibility is that the "hard" instances simply never occur in nature.[26] In both cases, $\mathcal{NP}$-completeness calls for a better understanding (e.g., refinement of the model at hand), a characterization of the instances for which the algorithm suggested by the model solves efficiently (and an argument that these conditions are consistent with what we see in nature).

This idea has caused some researchers to propose that our underlying conjecture, $\mathcal{P} \neq \mathcal{NP}$, should be viewed as a *law of nature*. Perhaps the first explicit such occurrence is this quote,[27] from Volker Strassen's laudation [Str86] for Les Valiant on his Nevanlinna Prize: *"The evidence in favor of Cook's and Valiant's hypotheses is so overwhelming, and the consequences of their failure are so grotesque, that their status may perhaps be compared to that of physical laws rather than that of ordinary mathematical conjectures."* Note that at that time, the utility of this mathematical conjecture to science was not as well understood as it is today. How to precisely articulate this mathematical statement as a relevant law of nature is of course still interesting to debate. An intuitive desire is to let it play the same role that the second law of thermodynamics plays in science: Scientists would be extremely wary to propose a model that violates it. One suggestion, by Scott Aaronson, is a stronger statement about the real world: *There are no physical means to solve $\mathcal{NP}$-complete problems in polynomial time.* A host of possible physical means that were actually attempted is discussed in [Aar05].

There is still the mystery of the ubiquitous presence of $\mathcal{NP}$-completeness in practically every subfield of CS, math, and essentially all sciences. With hindsight, this is an amplified (and much more relevant) incarnation of the ubiquity of undecidability in all these disciplines. Both are explained by the fact that computation, viewed (as above) as any process evolving via a sequence of simple, local steps, is so ubiquitous. Similarly, descriptions of properties of systems with many parts (either desired properties or observed properties, which are typically the outcomes of such computations) are often given or modeled by sets of simple, local constraints on small subsystems of the whole. As it happens, for almost all choices of constraints, their mutual satisfaction for given instances is undecidable if the system is infinite and is $\mathcal{NP}$-complete if finite. In much rarer cases, they lead to (respectively) decidable or polynomial-time solvable problems. Understanding these phenomena, and delineating the types of constraints across the tractable/intractable barrier, is an active field of study and will be discussed further in Section 4.3.

Concluding this somewhat philosophical section, we note another major impact of $\mathcal{NP}$-completeness. Namely, that it served as a role model for numerous other notions of computational universality. $\mathcal{NP}$-completeness turned out to be an extremely flexible and extendible notion, allowing numerous variants, which enabled capturing universality in other (mainly computational, but not only) contexts. It led to the definitions of classes of problems solvable using very different resource bounds. In most cases, these classes were also shown to have complete problems, capturing the difficulty of the whole class under natural reductions, with the benefits described above (some examples will be discussed in Section 4.1 and then in

---

[26]For example, it is quite possible that over billions of years of evolution, only proteins that are easily and efficiently foldable survived, and others became extinct.

[27]In this quote Cook's hypothesis is $\mathcal{P} \neq \mathcal{NP}$, and Valiant's hypothesis is what became known as $\mathcal{VP} \neq \mathcal{VNP}$ which we will discuss in Chapter 12.

later chapters). Much of the whole evolution of computational complexity, the theory of algorithms, and most other areas in theoretical computer science has been guided by the powerful approach of reduction and completeness. This progression has generated multiple theories of *intractability* in various settings and tools to understand and possibly curb or circumvent it (even though we are still mostly unable to prove intractability in most cases). The structures revealed by this powerful methodology send an important message to other disciplines.

I feel that the strongest impact of this message will be on the sciences. I will elaborate on this idea in Chapter 20 and summarize it here in one paragraph. The discussion in this chapter suggests integrating computational complexity aspects into *every* model of natural processes. In other words, scientists will come to account not only for the mechanism by which things evolve, and the way in which physical quantities are affected, but also for the amounts of relevant computational resources expended in that evolution. Doing so will add constraints that may guide scientists to better models, which nature can actually "carry out." When taking computational constraints into account in scientific models becomes standard practice, the way science is done will be revolutionized. While such a paradigm shift will take time to propagate across science, it is in the works! The above mentioned occurrence of $\mathcal{NP}$-completeness in so many scientific papers is just the beginning and illustrates both the importance of complexity to science and the willingness of scientists to embrace it. But beyond this, in more and more works (indeed, primarily collaborative works of computer scientists with scientists in other fields), we see models that integrate computational aspects into the description of natural processes and how this fusion can lead to radically new scientific insights. A few of the books, surveys, and articles representing this trend in various disciplines include [NRTV07, EK10, Val00, Kar11, HH13, Val13, CLPV14, Pap14], among many others. By way of example, [HH13], by physicists Daniel Harlow and Patrick Hayden, offers a complexity-theoretic based explanation that is possibly the only way out of the famous "black hole firewall paradox." Much more on these exciting developments can be found in Chapter 20.

4 Problems and classes inside (and around) $\mathcal{NP}$

This chapter touches on different aspects and variants of the $\mathcal{P}$ vs. $\mathcal{NP}$ question studied in computational complexity, the resulting complexity classes in the neighborhood of these two main classes, and the central questions about them. The first brief section lists a few types of problems that are not classification problems, leading mostly to classes containing $\mathcal{NP}$. The remaining sections are devoted to problems and classes in the potentially vast universe between the easiest and hardest problems in $\mathcal{NP}$, namely, between $\mathcal{P}$ and $\mathcal{NP}$-complete. We discuss degrees of intermediate complexities of problems in this universe, as well as *constraint satisfaction problems* (CSPs), for which there may be no such middle ground. The last two sections discuss the same universe from another perspective, motivated by average-case analysis of computational complexity as well as the more stringent computational needs of cryptography.

4.1 Other types of computational problems and complexity classes

There are many other types of computational problems that do not fall into the class $\mathcal{NP}$, arise naturally, and are studied intensively in both theory and practice. Some of the most natural types are:

- **Optimization problems.** Fix an $\mathcal{NP}$ problem and a cost function on solutions (witnesses). Given an input, find the *best* solution for it (e.g., find the largest clique, the shortest path, the minimum energy configuration). A natural relaxation of optimization problems asks for an approximation, namely to find a solution that is "close" to the best one. A complexity theory allowing the understanding of the limits of efficient approximation has been one of the most exciting developments in computational complexity, starting with the $\mathcal{PCP}$ Theorem 10.6. These problems are discussed in Sections 4.3 and 10.3.

- **Quantified problems.** A complete set for $\mathcal{NP}$ (which characterizes it via reductions) is SAT, namely, the set of all formulas F in variables x for which $\exists x\, F(x)$. Similarly, a complete set for co$\mathcal{NP}$ is the set of all formulas F in variables x for which $\forall x\, F(x)$. Generalizing from these examples, by allowing alternation of several quantifiers (and sets of variables), as is done, for example, in first-order logic, one obtains new complexity classes. For example, consider the set of formulas F in variables x, y satisfying $\exists x\, \forall y\, F(x, y)$, and use it to define a class (called Σ_2) of all problems efficiently reducible to it (if this reminds you of chess puzzles of the form "White to mate in 2 moves," you have the right intuition). The class Π_2 is similarly defined by formulas F satisfying $\forall x\, \exists y\, F(x, y)$. These naturally extend to the classes Σ_k, Π_k for any fixed natural number k (with $\mathcal{NP} = \Sigma_1$; co$\mathcal{NP} = \Pi_1$; and naturally, $\mathcal{P} = \Sigma_0 = \Pi_0$ when there are no quantifiers), with the obvious inclusions ($\Sigma_k \subseteq \Sigma_{k+1}$, $\Pi_k \subseteq \Pi_{k+1}$). While in first-order logic it is a theorem that each additional quantifier strictly adds descriptive power, the analog in computational complexity (e.g., $\Sigma_k \neq \Sigma_{k+1}$) is only a conjecture (extending $\mathcal{P} \neq \mathcal{NP}$). The complexity class that is the union over k of all these classes is called the *Polynomial Hierarchy*, denoted $\mathcal{PH}$. Its study was initiated by Stockmeyer [Sto76]. This class contains many natural problems, whose exact complexity is unknown. One of the most interesting and fascinating is *MCSP*, the minimum circuit size problem[1] whose status is surveyed in [AH17].

- **Counting problems.** Fix an $\mathcal{NP}$ problem. Given an input, find the *number* of solutions (witnesses) for it. Many problems in enumerative combinatorics and in statistical physics fall into this category. Here too, a natural relaxation of counting problems is approximation: computing a number that is "close" to the actual count. The natural home of most of these problems is a class called $\#\mathcal{P}$. A most natural complete problem for this class is $\#SAT$, which asks to compute the number of satisfying assignments of a given formula (more generally, counting versions of typical $\mathcal{NP}$-complete classification problems are $\#\mathcal{P}$-complete). A remarkable complete problem for it is evaluating the *Permanent* polynomial,[2] or equivalently, counting the number of perfect matchings of a given bipartite graph. Thus, even counting versions of easy classification problems (e.g., testing if a perfect matching exists) can be

---

[1] The input to this problem is a Boolean circuit, and the problem is to determine if there exists a smaller circuit computing the same function. We will formally define circuits in Section 5.2.

[2] A sibling of the Determinant, which is discussed in Chapter 12.

$\#\mathcal{P}$-complete. This discovery, the definition of the class $\#\mathcal{P}$, and the complexity theoretic study of enumeration problems originates from Valiant's papers [Val79a, Val79c]. A surprising, fundamental result of Toda [Tod91] efficiently reduces quantified problems (above) to counting problems (in symbols, $\mathcal{PH} \subseteq \mathcal{P}^{\#\mathcal{P}}$).

- **Strategic problems.** Given a (complete information, 2-player) game, find an optimal strategy for a given player. Equivalently, given a position in the game, find the best move. Many problems in economics and decision theory, as well as playing well the game of chess, fall into this category. The natural home for most of these problems is the class $\mathcal{PSPACE}$ of problems solvable using a polynomial amount of memory (but possibly exponential time). Indeed, many such games (appropriately extended to families of games of arbitrary sizes, to allow asymptotics, and restricting the number of moves to be polynomial in "board size") become complete for $\mathcal{PSPACE}$. This characterization of the basic memory (or space) resource in computation in terms of alternation of quantifiers (namely, as game strategies) arises as well from [Sto76] and obviously extends the bounded alternation games described above (which defined $\mathcal{PH}$). A major, surprising understanding of polynomial space is the result $\mathcal{IP} = \mathcal{PSPACE}$ of [Sha92]. It establishes $\mathcal{PSPACE}$ as the home of all problems having efficient *interactive proofs* (an important extension of "written proofs" captured by $\mathcal{NP}$), as discussed in Section 10.1.

- **Total $\mathcal{NP}$ functions.** These are search problems seeking to find objects that are *guaranteed* to exist (like local optima, fixed points, Nash equilibria) and are certified by small witnesses. In many such problems, the input is an *implicitly defined*[3] exponentially large graph, (possibly weighted, possibly directed). The task is to find a vertex with some simple property, whose existence is guaranteed by a combinatorial principle. For example, that every directed acyclic graph has a sink (and so the task is to find one), or that every undirected graph has an even number of vertices of odd degree (and so the task is, given one such vertex, to find another). In the paper initiating this study, Papadimitriou [Pap94] defines several complexity classes, each captured by one such principle. These classes lie between (the search problems associated with) $\mathcal{P}$ and $\mathcal{NP}$. One important example is the class $\mathcal{PLS}$, for polynomial local search, in which a complete problem is finding a local minimum in a weighted directed graph. Another is the class $\mathcal{PPAD}$, for which a natural complete problem is (a discrete version of) computing a fixed point of a given function. Computing the Nash equilibrium in a given 2-player game is clearly in this class, as the proof of Nash's theorem (that every game has such an equilibrium) follows simply from Brouwer's fixed-point theorem. A major result [DGP09, CDT09] was proving the converse: establishing that finding a Nash equilibrium is a complete problem for this class. These classes of problems and their complexity were studied in [BCE$^+$95] through the framework of proof complexity, a subject we will discuss in Chapter 6.

Figure 10 shows some of the known inclusions between these classes and some problems in them. Note that even though SAT and $CLIQUE$ are $\mathcal{NP}$-complete, while *Perfect Matching* is in $\mathcal{P}$, their counting versions are all in $\#\mathcal{P}$, and indeed, all three are complete for this class (*Permanent* is the counting problem for perfect matchings).[4]

I shall not elaborate on these families of important problems and classes here. Some of them will be mentioned in subsequent sections, but I will not develop their complexity theory systematically. Note that the methodology of efficient reductions and completeness illuminates much of their computational complexity in the same way as for classification problems.

4.2 Between $\mathcal{P}$ and $\mathcal{NP}$

We have seen that $\mathcal{NP}$ contains a vast number of problems, but that in terms of difficulty, nearly all of those we have seen fall into one of two equivalence classes: $\mathcal{P}$, which are all efficiently solvable, and $\mathcal{NP}$-complete. Of course, if $\mathcal{P} = \mathcal{NP}$, the two classes are the same. But assuming $\mathcal{P} \neq \mathcal{NP}$, is there anything else? Ladner [Lad75] proved the following result.

---

[3]For example, via a program computing the neighbors of any given vertex.
[4]We oversimplified the figure a bit; technically, we only know that $\mathcal{PH} \subseteq \mathcal{P}^{\#\mathcal{P}}$, rather than $\mathcal{PH} \subseteq \#\mathcal{P}$.
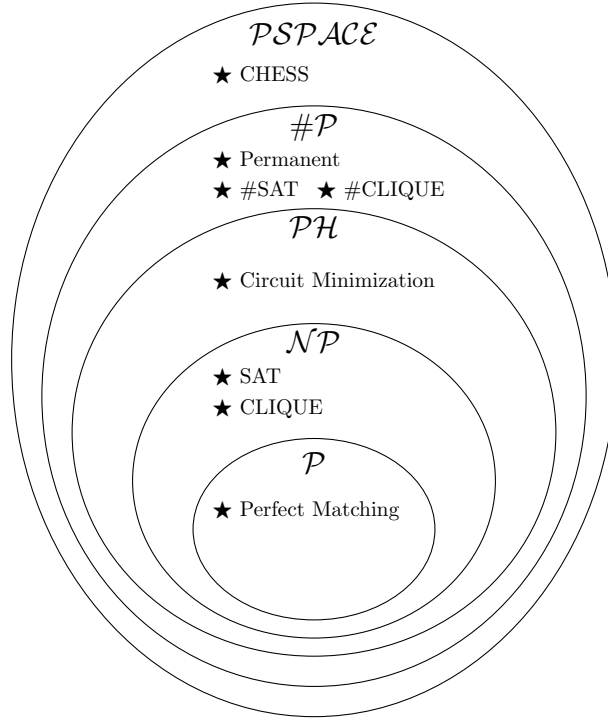
Figure 10. Between $\mathcal{P}$ and $\mathcal{PSPACE}$. As far as we know, all these classes may be equal.

Theorem 4.1 [Lad75]. *If* $\mathcal{P} \neq \mathcal{NP}$, *then there are infinitely many levels of difficulty in* $\mathcal{NP}$. *More precisely, there are sets* $C_1, C_2, \ldots$ *in* $\mathcal{NP}$ *such that for all* i, *we have* $C_i \leq C_{i+1}$, *but* $C_{i+1} \not\leq C_i$.

So, there is a lot of "dark matter" between $\mathcal{P}$ and $\mathcal{NP}$-complete. But are there any *natural* decision[5] problems that fall between these classes? We know only of very precious few candidates: those on the list below (some of which were also discussed in Section 3.5) and a handful of others. We discuss each in turn after listing them.

- **Integer factoring.** Given an integer, find its prime factors (a decision version might ask for the ith bit of the jth prime).

- **Stochastic games.** Three players, White, Black, and Nature, move a token on a directed graph, whose vertices are labeled with players' names. At every step, the token can be moved by the player labeling the vertex it occupies to another along an edge out of that vertex. Nature's moves are random, while White and Black play strategically. Given a labeled graph, and start and target nodes for the token, does White have a strategy that guarantees that the token reaches the target with probability $\geq 1/2$?

- **Knot triviality.** Given a diagram describing a knot (see, e.g., Figure 1), is it the trivial knot?

- **Approximate shortest lattice vector.** Given a (basis for a) lattice L in $\mathbb{R}^n$ and an integer k, does the shortest vector of L have (Euclidean) length at most k, or at least kn? (It is guaranteed that this minimum length is not in $[k, kn]$.)

- **Graph isomorphism.** Given two graphs, are they isomorphic? Namely, is there a bijection between

---

[5]We discussed search problems in this gap in Section 4.1.

their vertices that preserves the edges?[6]

- **Circuit minimization.** The notions appearing in this problem description will be formalized in Chapter 5. Intuitively, it asks for the fastest program computing a function on fixed-size inputs. More formally, given a truth table of a Boolean function f and an integer s, does there exist a Boolean circuit of size at most s computing f? Some evidence of the "intermediate status" of this problem can be found in [AH17] and its references.

Currently we cannot rule out that efficient algorithms will be found for any of these problems, and so some may actually be in $\mathcal{P}$. But we have good formal reasons to believe that they are not $\mathcal{NP}$-complete. This is interesting; we already saw that if a problem is $\mathcal{NP}$-complete, it is an indication that it is not *easy* (namely, in $\mathcal{P}$), if we believe that $\mathcal{P} \neq \mathcal{NP}$. What indications do we have that a problem is *not* universally *hard* (namely, that it is not $\mathcal{NP}$-complete)? Well, if, for example, the problem is in $\mathcal{NP} \cap \text{co}\mathcal{NP}$, and we believe $\mathcal{NP} \neq \text{co}\mathcal{NP}$, then that problem cannot be $\mathcal{NP}$-complete. In both arguments above, unlikely collapses of complexity classes ($\mathcal{P} = \mathcal{NP}$ and $\mathcal{NP} = \text{co}\mathcal{NP}$) give us (perhaps with different confidence levels) an indication as to the complexity of specific problems. This is somewhat satisfactory, in the absence of a definite theorem about their complexity. In particular, we can now explain better why the problems above are not likely to be $\mathcal{NP}$-complete.

The first four problems are all in $\mathcal{NP} \cap \text{co}\mathcal{NP}$. This is clear for (the decision problem of) factoring. For stochastic games, this result is proved in [Con93]. The lattice problem was resolved in [AR05]. The knottedness problem is special: It is a rare example where both inclusions are highly nontrivial. Membership in $\mathcal{NP}$ was proved in [HLP99] (and again, very differently, in [Lac15]). Membership in $\text{co}\mathcal{NP}$ was first proved conditionally on the generalized Riemann hypothesis (GRH) in [Kup14],[7] and only recently a different proof that requires no unproven assumption was given in [Lac16].

Graph isomorphism, while in $\mathcal{NP}$, is not known to be in $\text{co}\mathcal{NP}$, and so we cannot use the same logic to rule out its possible $\mathcal{NP}$-completeness directly. However, one can apply very similar logic. Graph nonisomorphism has a different type of short, efficient proof, called *interactive proof*, discussed in Chapter 10. Using this, one can prove that if graph isomorphism is $\mathcal{NP}$-complete, it would yield a surprising collapse of the polynomial time hierarchy $\mathcal{PH}$ (defined in Section 4.1). Of course, with the recent quasi-polynomial-time algorithm for graph isomorphism [Bab15] mentioned above, we have much better reasons to believe that it cannot be $\mathcal{NP}$-complete.

The last problem, circuit minimization (which has several variations), is even more mysterious than the previous four. Numerous papers have been written on "unlikely" consequences of its possible easiness (being in $\mathcal{P}$) and its possible hardness (being $\mathcal{NP}$-complete). A recent survey on the topic is [All17].

Finding other natural examples (or better yet, classes of examples) like these will enhance our understanding of the gap $\mathcal{NP} \setminus \mathcal{P}$. Considering the examples above, we expect that mathematics is a more likely source for them than, say, industry. However, for some large classes of natural problems, we know or believe that they *must* exhibit this dichotomy: Every problem in the class is either in $\mathcal{P}$ or is $\mathcal{NP}$-complete. These are classes of constraint-satisfaction problems, which we discuss next.

4.3 Constraint satisfaction problems (CSPs)

This section contains, besides extremely general and central conjectures of algorithms and complexity, also the story of persistence over decades and surprising mathematical sources needed to make progress towards conquering these problems.

4.3.1 Exact solvability and the dichotomy conjecture

A host of natural problems can be cast as satisfying a large collection of constraints on a set of variables. Solving a system of linear equations or polynomial equations over some field, satisfying a Boolean formula,

---

[6]The recent breakthrough of Babai [Bab15] gives a quasipolynomial-time algorithm for this problem (namely of complexity roughly $\exp((\log n)^{O(1)})$), bringing it very close to $\mathcal{P}$. See also the exposition [HBD17] of this result.

[7]It may seem mysterious what the GRH has to do with knots, and I encourage you to look at the paper to find out.

and coloring a graph are a few of many examples. Here we will be interested in *local* and *uniform* collections of constraints. "Locality" means that each constraint is on a constant number of variables. "Uniformity" means that all constraints are of the same type. For example, in a 3-coloring of planar maps, the variables correspond to the intended colors of the regions (each can take one of three values, say, Red, Green, Blue), and each pair of adjacent regions dictates one constraint: that the colors of these two regions must be different.

Let us give a more formal definition. Fixing *arity* k (the locality parameter), alphabet Σ (possible values for the variables), and a relation $R \subseteq \Sigma^k$ (defining the set of tuple values satisfying the constraint), we denote by $\mathrm{CSP}(k, \Sigma, R)$ the following computational problem. Given a collection of k-tuples from a set of n variables, is there an assignment of the variables from Σ^n that satisfies *all* constraints (namely, the value assigned to each given tuple is in the relation R)? These are CSPs with one relation. More generally, we can allow several relations instead of one, and we refer to all as "CSPs."

For example, for graph 3-coloring, there is a single (2-ary) relation ($a \neq b$) over the alphabet $\{1, 2, 3\}$ of three colors (formally, this relation contains the three pairs $R_{3col} = \{(1, 2), (1, 3), (2, 3)\}$). An instance of the problem $\mathrm{CSP}(3, \{1, 2, 3\}, R_{3col})$ is a collection of such constraints ($x_i \neq x_j$) over n variables that can take any of these three values. The pairs appearing in the constraints are the edges in this n-vertex graph. For another example, consider the $3 - SAT$ problem of Boolean satisfiability. Here one needs to introduce eight 3-ary relations, one for each of the eight literal combinations of negating or not the variables in a single clause (e.g., $(a \lor b \lor c)$, $(\bar{a} \lor b \lor c)$). An instance of this problem is a collection of constraints of this form over n variables x_i taking Boolean values in $\{0, 1\}$. Namely, it is a standard $3 - SAT$ instance.

A bold conjecture of Feder and Vardi [FV98], called the *Dichotomy conjecture*, asserts that for this vast set of problems, there are no intermediate levels of complexity of the type of Ladner's theorem from Section 4.2—each CSP is either in $\mathcal{P}$ or else is $\mathcal{NP}$-complete.

Conjecture 4.2 (Dichotomy conjecture [FV98]). Every CSP is either in $\mathcal{P}$ or is $\mathcal{NP}$-complete.

This conjecture was proved by Schaefer [Sch78] for the subclass of all CSPs on binary alphabets. His proof actually characterizes which constraints (namely, relations) give rise to easy CSPs and which yield hard ones. Further work by Bulatov and Jeavons on the general case [BJ01] sought such characterization in terms of a certain algebraic property of the relations defining each particular CSP. This property of the underlying relation is having non-trivial *polymorphisms*: these are functions combining finite tuples of sequences in a relation R to another sequence in the relation R (see the survey [BKW17] for much more).[8] Remarkably, the existence of non-trivial polymorphisms can distinguish, for example, between relations like *linear equations* and *disjunctions*, making the first CSP easy (in $\mathcal{P}$) and the second hard ($\mathcal{NP}$-complete). Observe that such characterizations require developing meta-algorithms in the easy case, and meta-reductions for proving hardness, as they derive each type of result simultaneously for infinitely many CSPs. This algebraic program was carried out for relations on ternary alphabets by Bulatov [Bul06], proving the conjecture in this case as well.

As this book went to print, proofs of the full dichotomy conjecture were announced by Rafiey, Kinne and Feder [?] and by Bulatov [Bul17] and Zhuk [Zhu17]! We state it as a theorem, proving the conjecture above. The remarkable *algebraic* theory which needed to be developed for this major algorithmic result is perhaps a lesson about how diverse the sources of efficient algorithms (and reductions) can be.

Theorem 4.3 (Dichotomy theorem [Bul17, Zhu17]). *Every CSP is either in $\mathcal{P}$ or is $\mathcal{NP}$-complete.*

There are a variety of extensions of CSPs and questions about them. In Section 11.2 we will discuss *quantum Hamiltonians*, which constitute a quantum analog of the above "classical" CSPs. They arise naturally in physics, and give rise to natural computational complexity questions and results about algorithms, completeness, proofs, approximations, dichotomy, and so forth.

Back to classical CSPs, there is major interest in *counting* the number of satisfying assignments (rather than the existence of one). Such questions are central e.g. in algebraic combinatorics and statistical

---

[8]To me polymorphisms are very reminiscent of the combinations of tuples of accepting computations to a new one in the *Fusion Method*, described in [Wig93] and in Section 5.2.4 on Natural Proofs of circuit lower bounds. But I know of no formal connection.

physics. An extensive theory was developed in which strong dichotomy theorems (now between $\mathcal{P}$ and $\#\mathcal{P}$-completeness) can be proved—see the results and historical account in the paper by Cai and Chen [CC12] and their book [CC17]. We will discuss in some detail the question of solving CSPs *approximately*.

4.3.2   Approximate solvability and the unique games conjecture

A very natural question to ask about CSPs is how efficiently they find good *approximate* solutions, namely, assignments to the variables that satisfy many of the given constraints. For example, in the graph 3-coloring problem mentioned above, it is very easy to satisfy 2/3 of the edge constraints (note that a random coloring would do so on average).[9] We also know that satisfying all of them is $\mathcal{NP}$-hard. What is the best *approximation ratio* achievable efficiently? For example, can we get 99%?

The huge mathematical field of optimization studies such questions (not only for CSPs), seeking efficient algorithms that produce good approximate solutions to problems for which the optimum is hard to find. For decades it was not known how to argue hardness (say, $\mathcal{NP}$-hardness) of approximation problems as the one above for 3-coloring. This changed dramatically with the invention of probabilistically checkable proofs (PCPs), and the revolutionary PCP Theorem 10.6, discussed in Section 10.3. This theorem and subsequent developments enabled not only proving such hardness results for many problems, but for some even pinpointing *precisely* the limits of efficient approximation. For example, while satisfying 1/2 fraction of a given set of linear equations modulo 2 is in $\mathcal{P}$ (find such an algorithm!), satisfying $1/2 + \epsilon$ fraction (for any $\epsilon > 0$) turns out to be $\mathcal{NP}$-hard.

That there is such a sharp transition in complexity for this problem may seem extremely surprising. However, a similar dichotomy conjecture says that for *every* CSP, this is indeed the case. The truth of this conjecture seems to hinge on the following remarkable computational problem. Its $\mathcal{NP}$-completeness status (is it, or is it not?), and more generally, its exact computational complexity have been both the most intriguing and most important problems in the past decade in this field. This problem was suggested by Khot [Kho02], who called it (for a reason) the *Unique Games* problem. I now explain the problem and touch on why it is so central. Section 10.3 will discuss the surprising origins of this problem.

Unique Games: Fix $\epsilon > 0$ and integer m. The problem $\text{UG}(\epsilon, m)$ is the following. The input is a system of linear equations in n variables $x_1, x_2, \ldots, x_n$ over $\mathbb{Z}_m$, with *two* variables per equation. An algorithm must answer "yes" if there is an assignment satisfying a fraction $1 - \epsilon$ of the equations, and answer "no" if no assignment satisfies more than a fraction ϵ of them (any answer is acceptable if neither of these is the case).[10] Observe that every unique games problem is a simple CSP (with m^3 relations, one for each $a, b, c \in \mathbb{Z}_m$, each a linear constraint on pairs of variables of the form $a \cdot x_i + b \cdot x_j = c$).

In his paper, Khot conjectured that this problem is $\mathcal{NP}$-complete. It is commonly called the unique games conjecture (UGC).

Conjecture 4.4 (UGC [Kho02]). *For every $\epsilon > 0$ there exists m such that $\text{UG}(\epsilon, m)$ is $\mathcal{NP}$-hard.*

Note that the UG problem may be viewed as an approximation problem—to solve the problem, it suffices to approximate the maximum number of satisfied equations to within a factor smaller than $(1 - \epsilon)/\epsilon$. This will distinguish the "yes" and "no" instances. Khot proved that improving certain approximation algorithms for some well-studied problems is at least as hard as solving the UG problem. Since his paper appeared, many more such reductions have been discovered. The unique games problem seems to be a new type of a complete problem, capturing limits of efficient approximation in many settings. This point was driven home powerfully when Raghavendra [Rag08] proved that, assuming UGC, there is a single, simple efficient meta-algorithm (based on semidefinite programming), which for every constraint satisfaction problem, achieves the best possible approximation ratio unless $\mathcal{P} = \mathcal{NP}$. Note that Raghavendra's result may be stated as a (conditional) dichotomy theorem.

Theorem 4.5 [Rag08]. *Assume UGC. Then for every CSP, there is a constant ρ such that approximating it to within approximation ratio ρ is in $\mathcal{P}$, but approximating it to any better ratio $\rho + \epsilon$ is $\mathcal{NP}$-hard for every $\epsilon > 0$.*

---

[9] Try finding an efficient deterministic algorithm that will produce such a 3-coloring.

[10] Such a problem is called a "promise problem," where algorithms can err on instances not satisfying the promise.

It is probably fair to say that, unlike most conjectures in this book, there is no consensus about the truth of UGC. Regardless, the study of UGC turned out to be a surprising source of problems in analysis, geometry, probability, and other fields. In particular, it was related by Raghavendra and Steurer [RS10] to expansion in graphs, a central topic we will discuss in Section 8.7 and other parts of this book. For more on this problem, the conjecture, and its connections, see [Kho10, O'D14]. And a few years later, the following happened, possibly providing stronger evidence to the truth of UGC.

2-2 Games: Khot's original paper [Kho02] on UGC also contains a somewhat weaker conjecture, on CSPs for which the constraints are linear equations as above, except that they have two possible values instead of one (i.e. have the form $a \cdot x_i + b \cdot x_j \in \{c_1, c_2\}$). Khot conjectured the $\mathcal{NP}$-hardness of this problem, and showed that even this yields great improvement of our understanding of many natural approximation problems. A rapid sequence of papers, culminating in [KMS18] (which describes this history) proves that the 2-2 Games problem is indeed $\mathcal{NP}$-hard! The complex proof in these papers requires the fine analysis of expansion properties of the so-called *Grassmann graph*.

4.4   Average-case complexity

It is important to stress that the worst-case analysis of algorithms, which we have adopted throughout (looking at the time to solve the worst input of each length), is certainly not the only interesting complexity measure (and not the only one studied). Often average-case analysis, focusing on "typical" complexity of algorithms, is far more interesting to study. After all, solving a hard problem most of the time may suffice in some applications. Thus, analyzing algorithms for given problems under *specific* natural input distributions (e.g., how they perform on average, or with high probability) is a large, important field, which was and is taken in earnest by the theory of algorithms and computational complexity. Some general approaches to modeling such "typical" distributions and developing algorithms for them include probabilistic analysis of algorithms [Kar76], semirandom models [FK01], smoothed analysis [ST04b] stable instances [BL12, BBG13], input distributions with certain moment bounds [HS17, KS17], and many others. We elaborate some on this effort and the challenge of understanding of heuristics in Section 20.7.2.

But the truth is that typically, in most practical applications, very little if anything is really known about the input distribution. Problem instances are generated by nature or people, and their distribution (or even support) is hard to pin down. To see this, consider the set of genomes and proteins processed by molecular biology algorithms, the set of signals from outer space processed by astrophysical algorithms, the set of search requests processed by Google, or even the set of mathematical structures that working mathematicians play with when contemplating a problem (e.g., the problems stated at the beginning of Chapter 2). Can one classify *all* natural input distributions? How should one construct a successful complexity theory for this average-case setting? Can all $\mathcal{NP}$-complete problems be easy on average? How should we formally define easy and hard problems in this distributional setting, and how do we compare the relative difficulty of distributional problems?

These highly nontrivial questions were first tackled by Levin [Lev86] and were better understood in his follow-up work with Impagliazzo [IL90]. Let us discuss some of the main definitions of Levin's theory and their subtleties. The reader will hopefully get a sense of the power of the methodology introduced earlier of classification, reductions, and completeness, as well as some difficulties that may arise when applying this methodology in new settings.

As input distributions are not known in advance, Levin generalizes the notion of a classification problem to *distributional* problems, whose description includes a distribution. These have the form (C, D), where $C \subseteq \mathbf{I}$ defines a property (or classification problem) as before, and D is a probability distribution on $\mathbf{I}$, according to which inputs are chosen. Next on the agenda is to define the class of *easy* distributional problems, a distributional analog of the class $\mathcal{P}$, which is called "dist$\mathcal{P}$," as those having fast algorithms "on average." This definition presents a significant challenge. Recall that the class $\mathcal{P}$ of easy worst-case problems enjoys strong robustness properties (e.g., it is invariant under polynomial changes to the model or to data representations). This desirable property is hard to guarantee in the distributional setting. Taking the obvious definition of average polynomial time—namely, when the expected run-time of an algorithm under the given distribution grows polynomially with input length—doesn't work. This expected value can

vary between polynomial and exponential for example, under quadratic change of input size or quadratic change of run-time per input. Levin overcomes this difficult by using a clever, nonstandard notion (which we skip here) of what it means for an algorithm to solve a problem C on input distribution D in average polynomial time. The class dist$\mathcal{P}$ of "easy" problems on average is then defined as the set of all distributional problems (C, D) possessing such an algorithm.

Next, with one eye toward applications of the theory, and another toward identifying *complete* distributional $\mathcal{NP}$ problems (that are as hard as all others), one has to judiciously choose the set of allowed input distributions to consider. It is not hard to see that to allow all distributions is hopeless. However, one would like the theory to include all *reasonable* distributions that can actually occur, like the natural and artificial examples mentioned above. The right choice turns out to be the efficiently sampleable distributions. A probability distribution is *efficiently sampleable* if it is the output distribution of any efficient algorithm that takes as input independent unbiased coin tosses (roughly, the uniform distribution on sequences of each given length). This is an extremely broad class of probability distributions! Indeed, assuming Nature does not perform computationally intractable tasks, this covers *all* distributions that algorithms will ever face. Once this choice is made, the distributional analog dist$\mathcal{NP}$ of $\mathcal{NP}$ can simply be defined as all pairs (C, D) with $C \in \mathcal{NP}$ and D an efficiently sampleable distribution.

It remains to define reductions and completeness. These are natural enough; the definition used in the worst-case setting suffices, as any mapping on instances $f : \mathbf{I} \to \mathbf{I}$ naturally extends to a mapping on distributions. But are there any complete problems, and even more, are there natural ones? Note that for (C, D) to be a complete problem, the distribution D must "capture" all other efficiently sampleable ones! This raises formidable technical challenges, which Levin overcomes, proving that a certain distributional version of the plane tiling problem is complete for dist$\mathcal{NP}$. Since Levin's original paper 30 years ago, precious few other reasonably natural complete problems have been found. It remains a challenge to exhibit a truly natural complete problem: perhaps one of the $\mathcal{NP}$-complete problems under the uniform distribution, or one of the problems arising in statistical mechanics (like the Ising model) under the natural Gibbs distributions.

Clearly, if $\mathcal{P} = \mathcal{NP}$, then dist$\mathcal{P} = $ dist$\mathcal{NP}$. Perhaps the most outstanding problem in this area is about the converse: do natural worst-case hard problems imply the existence of natural average-case hard ones?

Open Problem 4.6. Does $\mathcal{P} \neq \mathcal{NP}$ imply that dist$\mathcal{P} \neq$ dist$\mathcal{NP}$?

The reader can find more detail on this fascinating subject by Impagliazzo [Imp95b], and by Goldreich [Gol97].

4.5   One-way functions, trap-door functions, and cryptography

There is no better demonstration of the power of computational complexity ideas and methodology to completely transform the world we live in than the story of cryptography. I tell it very briefly here, focusing on the special types of "hard on-average" functions that underpin it. I address this story in far more detail in Chapter 18 of this book. There are numerous popular and technical texts that expound cryptography; I recommend Goldreich's book [Gol04] for a comprehensive technical development of its theoretical ideas.

Let us first motivate one-way functions describing the actual application that led to them, a password scheme from the 1960s due to Needham (described by Wilkes [Wil75, p. 91]). Believe it or not, access control has not changed much since then. A typical system asks the user for two pieces of information, *login* and *password*. Assume (without much loss in generality) that both are sequences of some length n, and that the login of user i is simply the number i. Every user secretly picks (at random or in any other way) a password x_i. Thus, if you type (i, x_i) (for any i whatsoever), then the system should let you in, and otherwise shouldn't. The main question is: How should the system store the passwords of its users? Certainly, it can store all pairs (i, x_i) in a protected system file (and whenever a user enters (i, z), check if $z = x_i$).

But even in the 1960s, hackers broke into systems, and so a better solution was sought. Needham suggested a way that will avoid altogether the need to hide the password file, as follows. Fix a function $f : \mathbf{I} \to \mathbf{I}$. The information stored in the password file would be pairs (i, y_i), with $y_i = f(x_i)$. Now let us see

what properties f should have to make this a good system. For one, it must be *easy to compute* f on any input x. After all, checking that (i, z) is legal now requires the system to check if $f(z) = y_i$. However, having access to the pairs (i, y_i) should help no one to figure out x_i for any i. In particular, f must be *hard to invert*: Given $f(x)$, it must be hard to obtain x (or any other pre-image of $f(x)$ if such exists). This should at least hold with high probability over a uniform random choice of x (and the system should recommend that users pick their passwords at random).

Thus, we arrived at the definition of *one-way* functions, put forth by Diffie and Hellman in their prescient, superbly written paper [DH76] as a foundation stone of their revolutionary theory of complexity-based cryptography. It is a function that is easy to compute but hard (on average) to invert.

Definition 4.7 (One-way function)**.** A function $f : \mathbf{I} \to \mathbf{I}$ is called *one-way* if $f \in \mathcal{P}$, but for every efficient algorithm A, its probability of computing any pre-image of f applied to a random input is small. Namely, for every (large enough) n,
$$\Pr[f(A(f(x))) = f(x)] \leq 1/2,$$
where the probability is taken over the uniform distribution of n-bit sequences x. Stated differently, the algorithm A is fed $y = f(x)$ for a random x, and should fail with high probability to produce any inverse of y.

Our choice of $1/2$ above as an upper bound on the inversion probability is arbitrary; as it happens, picking any bound in the range $[\exp(-n), 1 - 1/n]$ would yield an essentially equivalent definition; this is achieved via *amplification* of the success probability via repetitions—an idea we will meet in the sections on randomness, e.g., Section 7.1.

Let us meet Diffie and Hellman's suggestion (actually, they credit it to John Gill) for a one-way function, modular exponentiation, based on the assumed hardness of discrete logarithms.

Let p be a prime, g a generator of $\mathbb{Z}_p^*$, and define $\mathtt{ME}_{p,g} : \{1, 2, \ldots, p-1\} \to \{1, 2, \ldots, p-1\}$ by $\mathtt{ME}_{p,g}(x) = g^{x-1} \mod p$, the modular exponentiation function modulo p (and note that it is actually a permutation). Note that computing $\mathtt{ME}_{p,g}$ is easy on every input (via repeated squaring).[11] It is believed that for primes p for which $p - 1$ has few factors (e.g. $p - 1 = 2q$ with prime q), computing the inverse of $\mathtt{ME}_{p,g}$ (namely, the *discrete logarithm* modulo p) is exponentially hard,[12] even on average (many efforts to find better algorithms in the past decades have not refuted this belief). One can "glue" all these permutations (in several natural ways) into a single permutation $\mathtt{ME} : \mathbf{I} \to \mathbf{I}$, the *modular exponentiation* permutation, and conjecture:

Conjecture 4.8. The modular exponentiation function $\mathtt{ME}$ is a one-way function.

How is this conjecture related to the complexity classes we have already met? As it requires hardness on average, it implies that $\mathrm{dist}\mathcal{P} \neq \mathrm{dist}\mathcal{NP}$. Indeed, the existence of any one-way function will imply that! Moreover, as $\mathtt{ME}$ is a permutation, it implies that $\mathcal{NP} \cap \mathrm{co}\mathcal{NP} \neq \mathcal{P}$.[13] Indeed, any one-way permutation would imply this conjecture. These connections hopefully hint at the power of the computational complexity classification system in relating hardness of different problems whose complexity is unknown, both offering and limiting sources of examples of various types of hard problems.

Shortly after Diffie and Hellman's paper, Rivest, Shamir, and Adleman presented their candidate one-way function, modular powering, which is (indirectly) based on the assumed hardness of integer factoring. It has important advantages over modular exponentiation for cryptographic purposes that we will briefly discuss after defining it.

Let p, q be primes, $N = pq$, and c invertible modulo $\phi(N) = (p - 1)(q - 1)$. Define $\mathtt{MP}_{N,c} : \mathbb{Z}_N \to \mathbb{Z}_N$ by $\mathtt{MP}_{N,c}(x) = x^c \mod N$. This too is a permutation; if d is the inverse of c modulo $\phi(N)$, then $\mathtt{MP}_{N,d}(\mathtt{MP}_{N,c}(x)) = x$. Again, computing $\mathtt{MP}_{N,c}$ is easy. It is believed that inverting it is exponentially hard on random inputs, without access to the factors of N. Appropriately gluing all these functions into one

---

[11] Namely, computing all powers $g^{2^i} \mod p$ with $i \leq \log p$.

[12] As a function of the input length, which is roughly $\log p$ bits.

[13] As these classes are defined for decision problems, proving this requires a decision version of $\mathtt{ME}$. For example, given integers a, b, y, decide whether there is an x in the interval $[a, b]$ such that $\mathtt{ME}(x) = y$. As inverses exist and are unique, this classification problem is both in $\mathcal{NP}$ and $\mathrm{co}\mathcal{NP}$. Any algorithm for it can invert $\mathtt{ME}$ via binary search.

modular powering function (indeed, a permutation) $\text{MP} : \mathbf{I} \rightarrow \mathbf{I}$, we have a similar conjecture to the one for ME:

Conjecture 4.9. The modular powering function MP is a one-way function.

The significant advantage of this candidate one-way function is that it has a *trap-door*:[14] MP becomes easy to invert *if* you happen to have access to the factors p, q of N (because with the factors, one can efficiently compute the inverse d of c above). This property is the key behind the celebrated RSA *public-key cryptosystem*,[15] underlying most digital security systems since its invention. The way it works is extremely simple. I (and you, and Amazon; anyone, really) act as follows. I pick at random two large primes p, q and advertise (e.g., on my website) their product N and any c as above. If you want to send me a secret message x, you secretly encrypt x by computing $y = MP_{N,c}(x)$, and then send me y over any public channel (e.g., by e-mail). By Conjecture 4.9, no one can invert y without the factors of N, but I certainly can decrypt your message, as I have p, q. This is precisely how your credit card number is protected when you shop on-line; it is protected as long as a computational complexity assumption about the difficulty of integer factoring is true!

Let me spell out the absolutely remarkable content of the previous paragraph. The possible existence of trap-door one-way functions (which we will not formally define) does not merely enable on-line shopping and computer security (huge as the impact of these have been on society). It also allows any two parties, without any prior acquaintance, and in the presence of any others, to set up and use a secret language no one can understand! We note that in an "information theoretic" setting, when computation is free, such a feat is patently impossible. The basic premises of computational complexity—limits on computing power and the existence of natural hard functions—are absolutely essential.

And this was just the beginning. Trap-door functions, born to solve the problem of secret communication, turned out to solve practically every cryptographic problem imaginable. After cryptography was set up on formal foundations in the seminal paper of Goldwasser and Micali [GM84], and statements such as the above about secrecy and privacy could be mathematically formulated and proved, the crazy 1980s exploded with papers on other "impossible" tasks becoming possible. These showed that on the basis of trap-door functions, such tasks as *contract signing, secret exchange, playing poker over the telephone, oblivious computation* and many others could rest, culminating in the general protocols of [Yao86,GMW87]. Moreover, even the weaker one-way functions were found to be sufficient for, and indeed computationally equivalent to, such notions as *private-key cryptosystems*, *pseudo-random generation* (more on that in Section 7.3) and *zero-knowledge proofs* (more on that in Section 10.2).

It is hard to do justice in a few pages to the new levels of intricacy and subtlety introduced by the needs of cryptography to complexity-theoretic models and notions. We will discuss these modeling issues and results in much greater length in Chapter 18 and keep our discussion here rather brief. For one thing, most problems involve two or more parties, introducing interactive *protocols* instead of single-party *algorithms*. The importance of adversaries and adversarial thinking was taken far beyond worst-case analysis (where all an adversary can do is choose bad inputs) to the near-arbitrary abuse that cryptographic protocols must resist from parties who do not follow them. Reductions between cryptographic protocols and primitives were required to preserve, beyond efficiency, properties like *knowledge*, and were allowed to manipulate algorithms in new ways, far beyond applying them to given inputs (e.g., to rewind them again and again from arbitrary states). This has enriched computational complexity in a great many ways. One huge impact we will see in Chapter 7 is on a fresh understanding of *randomness*, a concept studied over millennia by many disciplines.

Diffie and Hellman wrote their paper soon after the birth of computational complexity and the definitions of $\mathcal{P}, \mathcal{NP}$, and $\mathcal{NP}$-completeness. In these early days, there was optimism that $\mathcal{P} \neq \mathcal{NP}$ would soon be proved, and they had every reason to expect that the hardness on average and one-wayness of functions could be proved unconditionally. As we know, this did not pan out, and we have to be content with *candidate* one-way and trap-door functions and with finding other means to support their assumed hardness.

---

[14] A notion sometimes signifying private or secret access.

[15] This notion was already suggested in [DH76], which also explained how it can be indirectly obtained (via a *key exchange* protocol) from the one-way function ME. Later, El Gamal [Elg85] showed how to obtain a public-key cryptosystem directly from ME.

For worst-case complexity, the abundance of $\mathcal{NP}$-complete problems of practical importance, and the huge (and independent) efforts invested over decades in trying to efficiently solve them, lends confidence to the assumption that they are hard. For one-way functions, Levin [Lev87] constructed a *complete* one-way function, namely, a function that is one way if one-way functions exist at all. However, it is not quite natural, and no one would dream of using it in actual cryptosystems.

So, what the world has to rely on to keep enjoying the benefits of cryptography is the assumed hardness of individual problems, such as the discrete logarithm and integer factoring. Of course, since they are used in practically all computer security systems, enormous efforts were made (by good and bad people) to find fast(er) algorithms for them, but so far, nothing reasonably efficient has been found. Attempts to come up with alternatives have also occupied cryptographers. For one-way functions, we actually have plenty of candidates (indeed, almost any computer program probably computes a one-way function, though it is unclear how one would prove that obtaining the input from the output is hard). In contrast, for trap-door functions (on which public-key encryption is based), we have precious few other examples besides integer factoring. A prominent candidate, that is very different from the number-theoretic problems above, is due to a sequence of works [Ajt96, AD97, PW11]. Here security is based on the hardness of finding short vectors in random lattices, a problem we discuss in Section 13.8. Having more candidate problems of trap-door functions is crucial! We will see in Chapter 11 on quantum computation, that both the discrete logarithm and integer factoring problems have fast *quantum* algorithms. So, if quantum computers are ever built, current security systems will become obsolete, whereas no such efficient quantum algorithms are known (yet) for these lattice problems.

But for all we know, there may be a fast classical algorithm for factoring (and some number theorists strongly believe this to be the case). Perhaps the most important problem in placing cryptography on solid foundations is to positively answer the following problem.

Open Problem 4.10. Does $\mathcal{P} \neq \mathcal{NP}$, (or even $\mathrm{dist}\mathcal{P} \neq \mathrm{dist}\mathcal{NP}$) imply the existence of one-way functions (and even trap-door functions)?

Proving hardness (and its difficulties) is the subject of the next chapter.

5 Lower bounds, Boolean circuits, and attacks on $\mathcal{P}$ vs. $\mathcal{NP}$

To prove that $\mathcal{P} \neq \mathcal{NP}$, we must show that for a given problem, no efficient algorithm exists. A result of this type is called a *lower bound* (limiting from below the computational complexity of the problem). Several powerful techniques for proving lower bounds have emerged in the past decades. They apply in two (very different) settings. I now describe both and explain why they seem to stop short of proving $\mathcal{P} \neq \mathcal{NP}$. I only mention very briefly the first, diagonalization, and concentrate on the second, Boolean circuits.

Note that Boolean circuits are studied as a computational model in their own right, not only in the context of lower bounds. The connections between this, so-called nonuniform model of circuits, to the usual uniform model of algorithms (e.g., Turing machines) are not completely understood. The main focus on circuits for lower bound attempts is that each circuit is a finite object, and one can hope (and succeed in limited cases, as we shall see) to analyze them using combinatorial methods.

5.1 Diagonalization and relativization

The diagonalization technique goes back to Cantor and his argument that the cardinality of the real numbers is larger than the cardinality of the integers. Diagonalization was used by Gödel in his incompleteness theorem, and by Turing in his undecidability results (which the reader may wish to recall). It was then refined to prove computational complexity lower bounds on computable functions. A typical theorem in this area is the *time-hierarchy* theorem of Hartmanis and Stearns [HS65], which essentially says that more time buys more computational power. For example, there are functions computable in time n^3, say, which are not computable in time n^2. The heart of such arguments (scaling down Turing's undecidability proof) is the existence of a "universal algorithm," which can simulate every other algorithm with only a small loss in efficiency. Thus an n^3-time machine can simultaneously diagonalize against *all* n^2-time algorithms and compute a function that disagrees with each of them on some input. More sophisticated uses of diagonalization yield other important lower bounds on Turing machines (e.g., see [PPST83, For00]).

Can such arguments be used to separate $\mathcal{P}$ from $\mathcal{NP}$? This depends on what we mean by "such arguments." The important paper by Baker, Gill, and Solovay [BGS75] proposes a formalization of this proof technique and shows that with this formalization, no such separation can be obtained. This is the first in a sequence of papers that aims to explain the limitations of common proof techniques. Their argument has two parts. First, they note a common feature shared by many diagonalization-like proofs of complexity results, called *relativization*. A proof relativizes if it remains valid when we modify all algorithms involved (both the simulated and the simulating machines) by giving them free ability to solve instances of any fixed (but arbitrary) problem $S \subseteq \mathbf{I}$. In the common jargon, these machines are all allowed free access to an "oracle" that answers questions about membership in S.[1] In the example above, an n^3-time universal machine with access to S can simulate every n^2-time machine with similar access, and so can diagonalize against it as well as in the original proof. The second part of the [BGS75] paper then shows that relativizing arguments do not suffice to resolve the $\mathcal{P}$ vs. $\mathcal{NP}$ question. To show this, these authors define two different oracles, say, S' and S'', whose presence elicits opposite answers to the $\mathcal{P}$ vs. $\mathcal{NP}$ question. Namely, with access to S', the power of "guessing" of $\mathcal{NP}$-machines magically disappears, yielding "$\mathcal{P} = \mathcal{NP}$", while with access to S'', the power of guessing becomes provably exponentially more powerful, yielding "$\mathcal{P} \neq \mathcal{NP}$". Relativization is further discussed by Fortnow [For94].

Even today, decades later, most complexity results do relativize. A major way to prove nonrelativizing results follows from the *arithmetization* technique discussed in Section 10.1 on interactive proofs. This was used to prove some lower bounds (e.g., [Vin04, San09]) that do *not* relativize. To curb the power of this technique, Aaronson and Wigderson [AW09] defined *algebrization*, a generalization of relativization that incorporates proofs that use arithmetization. They show that proofs that algebrize are still too weak to resolve the $\mathcal{P}$ vs. $\mathcal{NP}$ question, as well as other complexity challenges.

---

[1] S can be arbitrarily hard, even undecidable.

5.2 Boolean circuits

Boolean circuits are another basic model of computation, which we now explore. An excellent text on this subject is by Jukna [Juk12].

A *Boolean circuit* may be viewed as the hardware analog of an algorithm (software). Indeed, it abstracts the integrated circuits inside real computers and many physical control devices. Computation of a circuit on the Boolean inputs proceeds by applying a sequence of Boolean operations (called *gates*) to compute the output(s). Here we will consider the most common *universal* set of gates (sometimes called the *de Morgan* basis), $\{\wedge, \vee, \neg\}$: logical AND (conjunction), OR (disjunction), and NOT (negation), respectively. We assume here that $\wedge, \vee$ are each applied to two arguments. Note that although an algorithm can handle inputs of any length, a circuit can only handle one input length (the number of input "wires" it has). Figure 11 illustrates a computation of the parity function on 4 bits; computation proceeds from the inputs (at the bottom) to the output (at the top).
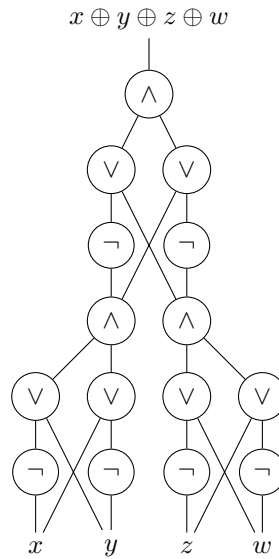
Figure 11. A circuit computing parity on 4 bits.

A circuit is commonly represented as a (directed, acyclic) graph, with the assignments of gates to its internal vertices. Note that a *Boolean formula*, commonly used in Boolean logic, is simply a Boolean circuit whose graph structure is a tree.

Recall that $\mathbf{I}$ denotes the set of all binary sequences, and that $\mathbf{I}_k$ is the set of sequences of length exactly k. If a circuit has n inputs and m outputs, it is clear that it computes a finite function $g\colon \mathbf{I}_n \to \mathbf{I}_m$. The efficiency of a circuit is measured by its *size*, which is the analog of time in algorithms.

Definition 5.1 (Circuit size). For a finite function g, denote by $\mathsf{S}(g)$ the size of the smallest Boolean circuit computing g.
More generally, for $f : \mathbf{I} \to \mathbf{I}$ with f_n the restriction of f to inputs of size n, we define $\mathsf{S}(f)$ to be the mapping from n to $\mathsf{S}(f_n)$.

As we care about asymptotic behavior, we will view functions $f : \mathbf{I} \to \mathbf{I}$ as a sequence of finite functions $f = \{f_n\}$, where f_n is a function on n input bits, namely, the restriction of f to input of size n. We shall study the complexity $\mathsf{S}(f_n)$ asymptotically as a function of n, and denote it by $\mathsf{S}(f)$. For example, let PAR be the parity function, computing whether the number of 1s in a binary string is even or odd. Then PAR_n is its restriction to n-bit inputs,[2] and it is not hard to check that $\mathsf{S}(PAR) = O(n)$.

---

[2]Namely, $PAR_n(x_1, x_2, \ldots, x_n) = x_1 \oplus x_2 \oplus \cdots \oplus x_n$.

Circuit families can efficiently simulate algorithms. It is quite straightforward to prove that an algorithm (say, a Turing machine) for a function f that runs in time T gives rise to a circuit family for the functions f_n of sizes $O((T(n))^2)$. The circuit c_n computing f_n simulates the given Turing machine on length-n inputs. As this infinite circuit family $\{c_n\}$ arises from a single algorithm, it is called a *uniform* family. Ignoring uniformity, we obtain the circuit analog of the algorithmic class $\mathcal{P}$.

Definition 5.2 (The class $\mathcal{P}/\text{poly}$). Let $\mathcal{P}/\text{poly}$ denote the set of all functions computable by a family of polynomial-size circuits, namely, all functions $f : \mathbf{I} \to \mathbf{I}$ such that $\mathsf{S}(f)$ grows polynomially with n.

The simulation above proves the following theorem.

Theorem 5.3. $\mathcal{P} \subseteq \mathcal{P}/\text{poly}$.

As a consequence of this simple simulation, lower bounds for circuits imply lower bounds for algorithms, and so we can try to attack the $\mathcal{P}$ vs. $\mathcal{NP}$ problem via circuits (completely dropping uniformity). The conjecture guiding this section, much stronger than Conjecture 3.6, is the following.

Conjecture 5.4. $\mathcal{NP} \not\subseteq \mathcal{P}/\text{poly}$.

Is this a reasonable conjecture? As mentioned above, $\mathcal{P} \subseteq \mathcal{P}/\text{poly}$. However, the converse of this statement fails badly! There exist *undecidable* functions f (which cannot be computed by Turing machines at all, regardless of their running time), that have linear-size circuits.[3] This extraordinary power of small circuits to solve undecidable problems comes from dropping the uniformity assumption; the fact that circuits for different input lengths share no common description. Indeed, the circuit model is sometimes called "nonuniform."

If small circuits can compute undecidable functions, this seems to make proving super-polynomial circuit lower bounds a much harder task than proving that $\mathcal{P} \neq \mathcal{NP}$. However, there is a strong sentiment that the extra power provided by nonuniformity is irrelevant for problems in $\mathcal{NP}$. This sentiment is supported by a theorem of Karp and Lipton [KL82], proving that the (nonuniform) assumption $\mathcal{NP} \subseteq \mathcal{P}/\text{poly}$ implies a surprising "collapse" of (uniform) complexity classes capturing *quantified problems* (defined in Section 4.1). Such a collapse is similar to, but weaker than, the statement $\mathcal{NP} = \text{co}\mathcal{NP}$.

Theorem 5.5 [KL82]. *If $\mathcal{NP} \subseteq \mathcal{P}/\text{poly}$, then $\Pi_2 = \Sigma_2$ (and hence, $\mathcal{PH} = \Sigma_2$).*

Still, what motivates replacing the Turing machine by the potentially more powerful circuit families when seeking lower bounds? The hope is that focusing on a *finite* model will allow for combinatorial techniques to analyze the power and limitations of efficient algorithms. This hope has materialized in the study of restricted classes of circuits (see, e.g., Section 5.2.3).

5.2.1 Basic results and questions

I have already mentioned several basic facts about Boolean circuits, and in particular the fact that they can efficiently simulate Turing machines. The next basic fact is that *most Boolean functions require exponential-size circuits*.

This is due to the gap between the number of functions and the number of small circuits. Fix the number of input bits, n. The number of possible functions on n bits is precisely 2^{2^n}. However, an upper bound on the number of different circuits of size s (via crudely estimating the number of graphs of that size and the choices for possible gates in each node) is roughly 2^{s^2}. Since every circuit computes one function, we must have $s > 2^{n/3}$ for *most* functions.[4] This is known as the *counting argument*, and is originally due to Shannon [Sha49b].

Theorem 5.6 [Sha49b]. *For almost every function $f : \mathbf{I}_n \to \{0, 1\}$, $\mathsf{S}(f) \geq 2^{n/3}$.*

---

[3]An example is $f = \{f_n\}$, where f_n is a constant function, whose output is 1 or 0, depending on whether n encodes a solvable Diophantine equation or not, respectively.

[4]In many cases, I will deliberately give weaker bounds than the best possible when it allows simpler calculations and crude estimates.

So hard functions for circuits (and hence for Turing machines) abound. However, as the hardness above is proved via a counting argument, it supplies no way of specifying one hard function. We shall return to the nonconstructive nature of this problem in Chapter 6. So far, we cannot prove such hardness for any *explicit* function f (e.g., for an $\mathcal{NP}$-complete function like SAT), even though it is believed to be true. It basically says that no significant time savings are possible over brute-force exhaustive search in solving SAT.

Conjecture 5.7. $\mathsf{S}(SAT) = 2^{\Omega(n)}$.

It is not surprising that we cannot prove this conjecture, as it is much stronger than Conjecture 3.6.[5] But our failure to establish lower bounds is much worse—no *nontrivial* lower bound is known for *any* explicit function. Note that for any function f on n bits (which depends on all its inputs), we trivially must have $\mathsf{S}(f) \geq n$, just to read the inputs. The main open problem of circuit complexity is beating this trivial bound for natural problems (say, in $\mathcal{NP}$)—after more than 60 years of intensive research, we still can't solve it.

Open Problem 5.8. Find an explicit function $f \colon \mathbf{I}_n \to \mathbf{I}_n$ for which $\mathsf{S}(f) \neq O(n)$.

A particularly basic special case of this problem is the question of whether addition is easier to perform than multiplication. Let ADD and $MULT$ denote, respectively, the addition and multiplication functions on a pair of integers (represented in binary). For addition, we have an optimal upper bound; that is, $\mathsf{S}(ADD) = O(n)$. For multiplication, the standard (elementary school) quadratic-time algorithm was greatly improved by Schönhage and Strassen [SS71] (via discrete Fourier transforms) to slightly super-linear, yielding $\mathsf{S}(MULT) = O(n \log n \log \log n)$. The best known algorithm is due to Fürer [Für09], but it is still slower than $n \log n$. Now, the question is *whether there exist linear-size circuits for multiplication.* In symbols, is $\mathsf{S}(MULT) = O(n)$?

Unable to prove any nontrivial lower bound, we now turn to restricted models. There has been some remarkable successes in developing techniques for proving strong lower bounds for natural restricted classes of circuits. We discuss in some detail two such models: first formulas, and then monotone circuits.

5.2.2   Boolean formulas

Formulas are prevalent throughout mathematics, mostly using arithmetic gates like $+$ and $\times$ (arithmetic computation is discussed in Chapter 12). We focus here on Boolean formulas, as are standard in logic, with the same set of de Morgan connectives $\{\wedge, \vee, \neg\}$ (for example, $(x \vee \neg y) \wedge z$). A formula may be viewed as a circuit having a tree structure. An example of a Boolean formula computing the parity function on 4 bits is shown in Figure 12. We denote the size of a formula by the number of occurrences of variables in it, namely, the number of leaves in the tree representing it (which, up to a factor of 2, is the same as the number of gates). Let us define the formula size of (necessarily one-bit output) Boolean functions.

Definition 5.9 (Formula size)**.** For a finite function $g : \mathbf{I}_n \to \{0, 1\}$, denote by $\mathsf{L}(g)$ the size of the smallest Boolean formula computing g. For $f : \mathbf{I} \to \{0, 1\}$, with f_n the restriction of f to inputs of size n, we define (as for circuits above) $\mathsf{L}(f)$ to be the mapping from n to $\mathsf{L}(f_n)$.

A formula is a universal computational model just like a circuit, in that every Boolean function can be computed by a Boolean formula. However, as we shall presently see, formulas are a weaker computational model, so we may hope to prove better lower bounds for it. Indeed, this already happens for essentially the simplest possible function, the *parity* function PAR discussed above. We mentioned that $\mathsf{S}(PAR) = O(n)$, and it is easy to prove (please try) that $\mathsf{L}(PAR) = O(n^2)$. Two of the earliest results in circuit complexity are lower bounds on parity. They use very different, important lower-bound techniques of wide applicability. Let us discuss both in turn.

Subbotovskaya [Sub61] proved that $\mathsf{L}(PAR) = \Omega(n^{1.5})$, inaugurating the *random restriction* method. How does this proof technique work to show that any formula for parity must be large (and more generally,

---

Figure 12. A formula computing parity on 4 bits.

that any computation C of a certain class computing a function f must be large)? First observe that fixing some of the input bits to some fixed constants results in a simpler computation C' (obtained after "hard-wiring" these values) computing the restricted function f' on the remaining variables. The idea now is that if the computational model is "weak" and the function is "complex," a clever choice of which variables to fix, and to which values, will render C' "trivial" and f' "nontrivial" to yield a patent contradiction.[6] As it happens, of course C is not given: We have to rule out any small computation, and so such a choice is not easy to make. Subbotovskaya's idea[7] is to simply choose such an input restriction at random! Here, such restrictions happen to *shrink* formulas at a much faster rate than the function. Thus, if C is too small, C' becomes a constant function, while f' becomes a parity function on the remaining unfixed variables, leading to the lower bound. We will soon return to this idea of shrinkage by random restrictions.

A decade later Khrapchenko [Khr71] improved the parity lower bound to a tight $\mathsf{L}(PAR) = \Omega(n^2)$. He used a natural induction on the formula structure to give a general lower bound in terms of the *sensitivity* of the function computed by it.[8] An alternative, *information-theoretic* proof of this lower bound was given by Karchmer and Wigderson [KW90]. Their *communication complexity* method shows that a formula for a given function may be viewed as a "communication protocol" for a related problem, so lower (and upper) bounds may be proved in this information-theoretic setting. This technique has further applications we will meet again below; it is explained in Section 15.2.3 after formalizing the communication complexity model in Chapter 15.

A new idea of Andreev [And87], pushed to its limit (using a tight analysis of the shrinkage of formulas under random restrictions) by Håstad [Hås98], gives the best known, nearly cubic gap. In yet another example of the mysterious connections across different subareas of computational complexity, Tal [Tal14] gives a different proof (with a slightly better bound) that surprisingly involves a *quantum argument*. More precisely, it uses a general simulation result of arbitrary Boolean formulas by quantum algorithms (we discuss quantum computation in Chapter 11).

Theorem 5.10 [Hås98, Tal14]. *There is a Boolean function f with $\mathsf{S}(f) = O(n)$ and $\mathsf{L}(f) = \Omega(n^{3-o(1)})$.*

The actual gap is believed to be exponential,[9] and proving a super-polynomial gap is a major challenge

---

[6]The words "trivial" and "nontrivial" have different meanings in different applications.

[7]Which was later rediscovered independently in [FSS84, Ajt83] in the context of *constant-depth* circuits, and then used throughout complexity theory. See [Bea94] for applications in circuit and proof complexity.

[8]Sensitivity is a key parameter (actually, a family of parameters) of Boolean functions, useful, for example, when viewed as voting schemes. See more in Section 13.7.

[9]Contrast this with the situation in *arithmetic* complexity to be discussed in Chapter 12, where formulas and circuits are much closer in power.

of circuit complexity.

Conjecture 5.11. There is a Boolean function f with $\mathsf{S}(f) = O(n)$ and $\mathsf{L}(f) \neq n^{O(1)}$.

One intuitive meaning of this conjecture (which we will not make formal) is asserting the existence of problems that have fast sequential algorithms but no fast parallel ones.

We now describe a particular approach to proving this important conjecture, due to Karchmer, Raz, and Wigderson [KRW95]. It calls for understanding how formula size behaves under the natural operation of function *composition*.

Definition 5.12. The composition $g \circ f$ of two Boolean functions $f : \{0,1\}^n \to \{0,1\}$ and $g : \{0,1\}^m \to \{0,1\}$ has mn input bits, viewed as m vectors $x^i \in \{0,1\}^n$, and is defined by $g \circ f(x^1, x^2, \ldots, x^m) = g(f(x^1), f(x^2), \ldots, f(x^m))$.

The most obvious formula computing this composition gives $\mathsf{L}(g \circ f) \leq \mathsf{L}(g) \cdot \mathsf{L}(f)$. The KRW conjecture [KRW95] is that there is no "significantly better" way to do so.[10]

Conjecture 5.13. For every f, g, $\mathsf{L}(g \circ f) \geq \alpha \cdot \mathsf{L}(g) \cdot \mathsf{L}(f)$ for some absolute constant $\alpha > 0$.

They further show how this conjecture (and even weaker ones) implies Conjecture 5.11, and they also lay out a program for proving it. For the status and history of progress of this program, see [GMWW14, DM16]. It is interesting that the cubic lower bound, Theorem 5.10, can be viewed as a proof of a very restricted form of Conjecture 5.13. Indeed, [DM16] provide an alternative proof of Theorem 5.10 via the communication complexity method.

Let's conclude this section with an alternative way to prove Conjecture 5.11, suggested by the communication complexity method. I will state it informally, and again refer the reader to Section 15.2.3. Consider the following task. I whisper in your ear an n-bit prime number x, and whisper in your friend's ear an n-bit composite number y. Your goal is to both agree on any number $z < 10n$, such that $x \neq y \pmod{z}$. Prove that this goal cannot be achieved by a conversation using $O(\log n)$ bits of communication, and you have proved Conjecture 5.11!

5.2.3 Monotone circuits and formulas

Many natural functions are *monotone* in a natural sense. Here is an example, from our list of $\mathcal{NP}$-complete problems. Let *CLIQUE* be the function that, given a graph on n vertices (say, by its adjacency matrix), outputs 1 iff it contains a *complete* subgraph of size (say) $\sqrt{n}$ (namely, all pairs of vertices in some size-$\sqrt{n}$ subset are connected by edges). This function is monotone, in the sense that adding edges cannot destroy any clique. More generally, a Boolean function is monotone if "increasing" the input (flipping input bits from 0 to 1) cannot "decrease" the function value (cause it to flip from 1 to 0).

A natural restriction on circuits comes by removing negation from the set of gates, namely, allowing only the gates $\{\wedge, \vee\}$. The resulting circuits are called *monotone circuits*, and it is easy to see that they can compute every monotone function.

A counting argument similar to the one we used for general circuits shows that most monotone functions require exponential-size monotone circuits. Still, proving a super-polynomial lower bound on an explicit monotone function was open for more than 40 years, until the invention of the *approximation method* by Razborov [Raz85a].

Theorem 5.14 [Raz85a], [AB87]. *CLIQUE requires exponential-size monotone circuits.*

Very roughly speaking, the approximation method replaces each of the $\{\wedge, \vee\}$ gates of the (presumed small) monotone circuit with other, judiciously chosen (and complex to describe) *approximating* gates, $\{\widetilde{\wedge}, \widetilde{\vee}\}$, respectively. The choice satisfies two key properties, which together easily rule out small circuits for *CLIQUE*:

---

[10]Indeed, the slack in this inequality can be super-constant, and it is interesting even if this holds for most functions f (or for most g).

(1) Replacing one particular gate by its approximator can only affect the output of the circuit on very *few* (in some natural but nontrivial counting measure) inputs. Thus, in a small circuit having a few gates, even replacing all gates by their approximators results in a circuit that behaves like the original circuit on most inputs.

(2) However, the output of *every* circuit (regardless of size) made of the approximating gates produces a function that disagrees with *CLIQUE* on *many* inputs.[11]

One natural view of the approximation method is as the description of a *progress measure* (or *potential function*) on circuits, to which each gate contributes only a little. If a function is costly in this measure, any circuit for it must be large. It is natural to view these small aggregate contributions of gates to this measure in a *dynamic* way, according to their order in the circuit. However, there is a *static* view of this method (dubbed the *fusion method*), which actually better captures the way lower bounds are proved, that is exposited in [Wig93].[12]

The *CLIQUE* function is well known to be $\mathcal{NP}$-complete, and it is natural to wonder whether small monotone circuits suffice for monotone functions in $\mathcal{P}$. However, the approximation method was also used by Razborov [Raz85b] to prove an $n^{\Omega(\log n)}$ size lower bound for monotone circuits computing the *perfect matching* problem (which is monotone and is in $\mathcal{P}$): Given a graph, can one pair up the vertices such that every pair is connected by an edge (see illustration in Figure 3)?

Theorem 5.15 [Raz85b]. *Perfect matching requires super-polynomial size monotone circuits.*

Interestingly, no exponential lower bound is known for monotone circuits for this problem. Communication complexity techniques (see below) were employed by Raz and Wigderson [RW92] to prove that perfect matching requires exponential size monotone formulas.

Theorem 5.16 [RW92]. *Perfect matching requires exponential size monotone formulas.*

Tardos [Tar87] finally proved an exponential gap between monotone and nonmonotone circuits for the problem of computing the (monotone, threshold version of the) Lovász' *Theta function* Θ. This problem is in $\mathcal{P}$ via semidefinite programming, and the lower bound uses that Theorem 5.14 is much stronger than stated: It holds even if the only input graphs G are either $k+1$-cliques (in which case, $\Theta(G) \geq k+1$) or complete k-partite graphs (in which case, $\Theta(G) \leq k$).

The relative power of circuits vs. formulas in the monotone case is also pretty well understood. The first separation was proved by Karchmer and Wigderson [KW90] for graph connectivity, a function which has simple monotone, polynomial-size circuits, but they showed requires $n^{\Omega(\log n)}$ size monotone formulas, a tight bound.[13]

Theorem 5.17 [KW90]. *Undirected graph connectivity (which has monotone polynomial-size circuits) requires super-polynomial-size monotone formulas.*

The monotone formula lower bounds in Theorems 5.16 and 5.17 were proved using the *communication complexity* method mentioned above (and explained in Section15.2.3).

The communication complexity method was generalized by Raz and McKenzie [RM99], giving much finer separations between monotone circuits and formulas. Their method (called the *lifting* or *pattern matrix* method in the literature) was greatly enhanced and better understood in a series of recent works, which allows applying it to other monotone models and obtaining stronger results more simply—see [RPRC16,PR17] and its historical discussions.

We conclude this section with something short monotone formulas *can* do: compute the Majority function MAJ. Even the task of constructing a nonmonotone polynomial-size formula for MAJ is not entirely trivial—please try it. However, whether MAJ has polynomial-size monotone formulas had been open for

---

[11]Indeed, such circuits can only compute small, monotone DNFs (namely, disjunctive normal form formulas).

[12]This paper also discusses how this view extends to computation itself, and surprising consequences it generated beyond lower bounds.

[13]A completely different way of proving a (weaker) super-polynomial separation follows the KRW conjecture [KRW95] (Conjecture 5.13), which in the monotone case becomes a theorem.

decades, until resolved in the early 1980s in two completely different ways, one by Ajtai, Komlós and Szemerédi [AKS83] and the other by Valiant [Val84a].

Theorem 5.18 [AKS83, Val84a]. *Majority has polynomial-size monotone formulas.*

The proof of Ajtai, Komlós, and Szemerédi is entirely constructive[14] but extremely complex, and it leads to a polynomial of exponent in the hundreds![15] In contrast, Valiant's proof is extremely simple and elegant and leads to a small polynomial bound, but it is only an existence proof. In what is one of the most stunning examples of the power of the probabilistic method (see [AS00]), he shows that while constructing small monotone majority formulas seems difficult, almost all of them do compute majority. The best known upper bound on the size is $O(n^{5.3})$. The best known lower bound (even on nonmonotone formulas) is $\Omega(n^2)$. We have no tools as yet to answer problems like the following.

Open Problem 5.19. Is there a monotone formula for majority of size $O(n^3)$?

5.2.4   Natural Proofs, or, Why is it hard to prove circuit lower bounds?

The 1980s saw a flurry of new techniques for proving circuit lower bounds on natural, restricted classes of circuits. Besides the *approximation method*, these include the *random restriction* method of Furst, Saxe, Sipser [FSS84], and Ajtai [Ajt83] (used to prove lower bounds on constant depth circuits, which we have not discussed), the *communication complexity* method of Karchmer and Wigderson [KW90] (used above for monotone formula lower bounds), and others. But they and all subsequent results fall short of obtaining any nontrivial lower bounds for general circuits, and in particular proving that $\mathcal{P} \neq \mathcal{NP}$.

Is there a fundamental reason for this failure? The same may be asked about any longstanding mathematical problem (e.g., the Riemann Hypothesis). A natural (vague!) answer would be that, probably, the current arsenal of tools and ideas (which may well have been successful at attacking related, easier problems) does not suffice.[16]

Remarkably, complexity theory can make this vague statement into a theorem. Thus, we have a formal excuse for our failure so far: we can classify a general set of ideas and tools, which are responsible for virtually all restricted circuit lower bounds known, yet must necessarily fail for proving general ones. This introspective result, developed by Razborov and Rudich [RR97], suggests a framework called *Natural Proofs*. Very briefly, a lower bound proof is *natural* if it applies to a *large, easily recognizable* set of functions. They first show that this framework encapsulates essentially all known circuit lower bounds. Then they show that natural proofs of general circuit lower bounds are unlikely, in the following sense. Any natural proof of a general circuit lower bound surprisingly implies, as a side-effect, a subexponential algorithm for inverting every candidate one-way function.

Specifically, a *natural* (in this formal sense) lower bound would imply subexponential algorithms for such functions as integer factoring and discrete logarithm, generally believed to be exponentially difficult (to the extent that the security of electronic commerce worldwide relies on such assumptions). This connection strongly uses *pseudo-randomness*, which will be discussed later (at the end of Section 8.4). A simple concrete corollary (see [RR97] for a more general statement) is that there is no natural proof that integer factoring requires circuits of size at least $2^{n^{1/100}}$ (the best current upper bound is $2^{n^{1/3}}$).

Does the difficulty of proving lower bounds arise from the choice of axioms underlying mathematics (e.g. as in the case of the famous *continuum hypothesis*)? This question is surveyed in [Aar03]. One interpretation of the work on the natural proofs framework is as an "independence result" from logical proof systems. Specifically, Razborov [Raz95b] initiated the formal study of the independence of proving general circuit lower bounds from a certain natural fragment of Peano arithmetic (this is related to the difficulty of proving propositional formulations of circuit lower bounds, discussed toward the end of Chapter 6 on *proof complexity*). How far up in terms of axiomatic logical power does this independence go? Note that it is possible

---

[14]Making essential use of *expander graphs*, which we will meet in Section 8.7.

[15]It solves a much more general problem: constructing a linear size, logarithmic depth sorting network, of which this result is a corollary.

[16]Such was the case, for example with Fermat's last theorem for centuries, during which tools developed to eventually allow Wiles and Taylor to prove it (and much more).

that the $\mathcal{P}$ vs. $\mathcal{NP}$ problem is independent from Peano arithmetic, or even ZFC set theory (as indeed is the continuum hypothesis). While such independence of $\mathcal{P}$ vs. $\mathcal{NP}$ is a mathematical possibility, few believe it today.

The "natural proof" framework was extended in several directions. These turned out to yield not only new barriers, and barriers on their power, but also new computational models (like *span programs*) and new connections (e.g. to the cryptographic problem of *secret sharing*). Many of these aspects are surveyed in [Wig93].

The lower bounds of Santhanam [San09] and Vindochandran [Vin04] mentioned in Section 5.1 are circuit lower bounds that bypass the natural proof barrier (as well as relativization). However, as we mentioned there, they both algebrize; see Aaronson and Wigderson [AW09]. The only circuit lower bound technique that avoids all known barriers originates with Williams (see [Wil14, AW18]). It uses a brilliant combination of diagonalization and simulation on the one hand, and circuit-complexity techniques[17] on the other. Unfortunately, this line of work has so far delivered relatively few lower bounds for rather weak circuit classes.

For proving super-polynomial *formula* lower bounds, Conjecture 5.13 on composition (discussed above in Section 5.2.2) seems to avoid all known barriers as well, and may thus be used to prove such lower bounds with currently available techniques. But it has yet to bear fruit.

Given the above discussion, one would certainly expect the problem of *computing* the minimum size circuit (say) for a given Boolean function to be computationally hard. One precise formulation of this computational problem, called *MCSP* (minimum circuit size problem) is the following: given as input a Boolean circuit, decide if there is a smaller one computing the same function. We don't even know if this problem is $\mathcal{NP}$-hard, and its computational complexity is an intensive research area (see [AH17] for a recent survey). Perhaps this (somewhat self-referential) study of *complexity of complexity* will assist us in proving lower bounds?

Concluding, I view the mystery of the difficulty of proving (even the slightest non-trivial) computational difficulty of natural problems to be one of the greatest mysteries of contemporary mathematics.

---

[17]Especially one from [IKW02], a result surprisingly based on *pseudo-randomness*! We will discuss interactions of circuit complexity and pseudo-randomness in Section 7.3.

6 Proof complexity

The reader will have seen, depending on experience, a variety of mathematical theorems and their proofs in various mathematical areas. However, it is quite likely that the focus this chapter takes on proofs, as well as the types of theorems considered here, are quite different than what you are used to. This fresh view of proofs is incredibly rich and brings out a beautiful set of mathematical problems and results. For extensive surveys on this material, see [BP98, Seg07, RW00], and in a bigger context, the books by Krajíček [Kra95, Kra19].

The concept of *proof* is what distinguishes the study of mathematics from all other fields of human inquiry. Proofs establish mathematical *theorems*, whose validity is independent of physical reality. While proofs are typically presented informally in mathematical papers and books, the confidence we have in their outcomes, the theorems, follows the knowledge (or belief) that they can be made completely rigorous—that is, formalized to a purely *syntactic* form whose absolute correctness can be easily verified. This verification algorithm is specified by a *proof system*, which determines for any two sequences of symbols x, y whether y is a proof of x. In most mathematical proof systems, y is a sequence of sound deductions deriving x from a set of "self-evident" axioms. Such a formalism makes proofs themselves the object of mathematical investigation, mainly in proof theory and logic.

Needless to say, the practice of mathematics is not a syntactic game. While rigor is paramount, mathematicians care about the *meaning* of what they prove and how they prove it. So theorems x would not be just abstract symbols, but rather typically describe meaningful properties of certain natural mathematical structures. Similarly, proof systems encompass various types of reasoning, and proofs y will typically combine ideas, techniques, and past theorems to "reason out" a new theorem. With centuries of experience, mathematicians often attribute (and argue about) such adjectives to proofs as "beautiful, insightful, original, deep" and, most notably, "difficult," which this section on proof complexity focuses on.

Is it possible to quantify, mathematically, the difficulty of proving various theorems? This is exactly the task undertaken in the field of *proof complexity*. It focuses on *propositional* proof systems (that we will soon define and discuss), which are the simplest from a purely logical standpoint; these systems are designed to prove statements about finite structures. Proof complexity seeks to classify propositional theorems (called "tautologies") according to the difficulty of proving them, much like circuit complexity seeks to classify functions according to the difficulty of computing them. Indeed, proof complexity bears a similar relation to *proof theory* as that of circuit complexity to *computability theory*. In proofs, just as in computation, there will be a number of models, called *proof systems*, capturing the power (or structure) of reasoning allowed to the verifier.

Proof systems exist in abundance in all areas of mathematics (and not just in logic), sometimes implicitly. Indeed, the statement that a given mathematical structure possesses some property is a canonical example of what mathematicians are trying to establish. And the mathematical *framework* for establishing those statements which are true—the theorems in such settings—can typically (and naturally) be viewed as a proof system. Let us consider some examples of proof systems outside the familiar logical ones (to which we will return later) and discuss the notion of proof *length* in each. These in particular reveal proof length as natural, and we will discuss this point further afterward. Some may not be familiar to you—feel free to skip or find out more yourself.

1. Hilbert's Nullstellensatz may be viewed as supplying a (sound and complete) proof system in which *theorems* are inconsistent systems of polynomial equations.[1] A *proof* expresses the constant 1 as a linear combination (with polynomial coefficients) of the given polynomials.

2. Each finitely presented group[2] can be viewed as a proof system, in which *theorems* are words that reduce to the identity element. A *proof* is the sequence of substituting relations that transforms the word to the identity. Such proofs have a nice geometric representation, called "Dehn diagrams" (or van Kampen diagrams), in which proof length is captured by the "area"—the number of regions in the diagram.

---

[1]Such a system may be viewed as a statement: "No valuation to the variables simultaneously satisfies all equations in the system."

[2]Namely, given by a finite set of generators and relations.

3. Reidemeister moves are a proof system in which *theorems* are plane diagrams of trivial (unknotted) knots. A *proof* is the sequence of moves reducing the given plane diagram of the knot into one with no crossings. Think of a circular string lying scrambled on the table, and the moves unscrambling it make local changes around one crossing at a time.

4. Von Neumann's minimax theorem gives a proof system for every zero-sum game. A *theorem* is an optimal strategy for White, and its *proof* is a strategy for Black that guarantees the same payoff.[3]

In each of these and many other examples, the *length* of the proof plays a key role, and the quality (or expressiveness) of the proof system is often related to how short the proofs it provides can be.

1. In the Nullstellensatz (over fields of characteristic 0), length (of the "coefficient" polynomials, typically measured by their degree and height) plays an important role in the efficiency of commutative algebra software (e.g., Gröbner basis algorithms).

2. The word problem in general is undecidable. For hyperbolic groups, Gromov's polynomial upper bound on proof length has many uses. One is his own construction of finitely presented groups with no uniform embeddings into Hilbert space [Gro03].

3. In a recent advance, a polynomial upper bound on the length of Reidemeister proofs for unknottedness was given in [Lac15].

4. In zero-sum games, happily, all proofs are of linear size.

We stress that the asymptotic viewpoint, namely, considering *families* of theorems as above and measuring their proof length as a function of the description length of the theorems proved, is natural and prevalent in mathematics. As is the case for computation, here, too, this asymptotic viewpoint reveals structure of the underlying mathematical objects, and economy (or efficiency) of proof length often correlates with better understanding them. While this viewpoint is relevant to a large chunk of mathematical work, it seems to fall short of explaining the difficulty of most challenges mathematicians face, namely, the difficulty of proving *single* statements (in which asymptotics are not present), such as the Riemann Hypothesis or $\mathcal{P} \neq \mathcal{NP}$.

Let us probe this point a bit deeper. As it turns out, often such "single" mathematical statements can be viewed and studied asymptotically (of course, this approach may or may not illuminate them better). For example, the Riemann Hypothesis has an equivalent formulation as a sequence of finite statements (e.g., about cancellations in the Möbius function) up to every finite bound n, which we shall see later in Section 8.3 on pseudo-randomness. Perhaps more interestingly, in Section 8.4, we will discuss a formulation of the $\mathcal{P}/\text{poly}$ vs. $\mathcal{NP}$ problem as a sequence of finite statements, whose proof complexity turns out to be strongly related to the natural proofs paradigm mentioned in Section 5.2.4 on the difficulty of proving circuit lower bounds. These are just examples of a general phenomenon: Statements in first-order logical systems, (like Peano arithmetic or fragments thereof) can be turned into a sequence of finite propositional statements, such that proof length lower bounds for them (in the appropriate propositional system) can imply unprovability in the first-order setting. In other words, the propositional setting we discuss here can provide standard unprovability and independence results! This idea was originated by Paris and Wilkie [PW85] for a particular fragment and was explained more generally by Krajíček in the book [Kra95].

All theorems that will concern us in this chapter are *universal* statements (e.g., an inconsistent set of polynomial equations is the statement that *every* assignment of values to the variables fails to satisfy them all). A short proof for a universal statement constitutes an equivalent formulation of that statement that is *existential*—the existence of the proof itself certifying the universal property (e.g., the existence of the "coefficient" polynomials in Hilbert's Nullstellensatz, which implies the inconsistency of the given polynomial equations). The mathematical motivation for this focus is clear—the ability to describe a property both universally and existentially constitutes *necessary* and *sufficient* conditions—a cornerstone of mathematical understanding, discussed in Section 3.5. Here we shall be picky and quantify that understanding according to a (computational) yardstick: the *length* of the existential certificate.

---

[3]Naturally, the reverse holds as well, and there is a duality between theorems and proofs in this system.

We shall restrict ourselves to *propositional* tautologies, namely, those ranging over Boolean variables (we will shortly see an example). These can naturally encompass all true statements about discrete structures and turn out to provide a remarkably broad and deep arena of study. This focus actually guarantees an exponential (thus a known, finite) upper bound on the proof length of any theorem considered, freeing us from any Gödelian worries of unprovability. It restricts the range of potential proof lengths (as with time in the case of $\mathcal{P}$ vs. $\mathcal{NP}$) to be between polynomial and exponential. In an analogous way to the computation time, exponential proof length here will correspond to trivial, "brute force" proofs and the possibility (or impossibility) of finding clever short proofs.

The type of statements, theorems, and proofs we shall deal with is best illustrated by the following example.

6.1   The pigeonhole principle—a motivating example

Consider the well-known "pigeonhole principle", stating that there is no injective mapping from a finite set to a smaller one. While trivial, we note that this principle was essential for the counting argument proving the *existence* of exponentially hard functions (Theorem 5.6)—this partially explains our interest in its proof complexity. More generally, this principle epitomizes *nonconstructive* arguments in mathematics, such as Minkowski's theorem that a centrally symmetric convex body of sufficient volume must contain a lattice point, or Erdős' probabilistic proof that a small Ramsey graph[4] exists. In these and many other examples, the proof does not provide any information about the object proved to exist. The same happens for proofs of other combinatorial tautologies that capture the essence of topological theorems (e.g., Brouwer's fixed point theorem, the Borsuk-Ulam theorem, and Nash's equilibrium)—see Papadimitriou [Pap94] for more.

Let us formulate the pigeonhole principle and discuss the complexity of proving it. First, we turn it into a sequence of finite statements. Fix $m > n$. Let PHP_n^m stand for the statement *there is no 1–1 mapping of m pigeons to n holes*. To formulate this mathematically, imagine an $m \times n$ matrix of Boolean variables x_{ij} describing a hypothetical mapping (with the interpretation that $x_{ij} = 1$ means that the ith pigeon is mapped to the jth hole).[5]

Definition 6.1 (The pigeonhole principle)**.** The pigeonhole principle PHP_n^m now states that

- either some pigeon $i \in [m]$ is not mapped anywhere (namely, all x_{ij} for a fixed i are zeros), or

- two pigeons are mapped to the same hole (namely, for some different $i, i' \in [m]$ and some $j \in n$, we have $x_{ij} = x_{i'j} = 1$).

These conditions are easily expressible as a formula using Boolean gates in the variables x_{ij} (called a *propositional formula*). Let us write it explicitly:

$$\left(\bigvee_{i \in [m]} \left(\bigwedge_{j \in [n]} \neg x_{ij} \right) \right) \vee \left(\bigvee_{i \neq i' \in [m]} \bigvee_{j \in [n]} (x_{ij} \wedge x_{i'j}) \right)$$

The pigeonhole principle is the statement that this formula is a *tautology* (namely, satisfied by *every* truth assignment to the variables).

Even more conveniently, the negation of this tautology (which is a *contradiction*, namely, a formula satisfied by no assignment) can be captured by a collection of constraints on these Boolean variables that are mutually contradictory. Such collections of constraints can be expressed in different languages:

- **Algebraic:** as a set of bounded degree polynomials over $\mathbb{F}_2$ (or other fields).

- **Geometric:** as a set of linear inequalities with integer coefficients (to which we seek a $\{0, 1\}$ solution).

---

[4]A graph without large cliques and independent sets.

[5]Note that we do not rule out the possibility that some pigeon is mapped to more than one hole—this condition can be added, but the truth of the principle remains valid without it.

- **Logical:** as a set of Boolean disjunctions.

We shall see in Section 6.3 that each setting naturally suggests (several) reasoning tools, such as variants of the Nullstellensatz in the algebraic setting, of Frege systems in the logical setting, and Integer Programming heuristics in the geometric setting. All of these can be formalized as proof systems that can prove this (and any other) tautology. Our main concern will be in the efficiency of each of these proof systems, and their relative power, measured in *proof length*. Before turning to some of these specific systems, we discuss this concept in full generality.

6.2   Propositional proof systems and $\mathcal{NP}$ vs. co$\mathcal{NP}$

Most definitions and results in this section come from the paper that initiated this research direction, by Cook and Reckhow [CR79]. We define proof systems and the complexity measure of proof length for each and then relate these to the complexity questions we have met already.

All theorems we shall consider will be propositional tautologies. Here are the salient features that we expect[6] from any proof system:

- **Completeness.** Every true statement has a proof.

- **Soundness.** No false statement has a proof.

- **Verification efficiency.** Given a mathematical statement T and a purported proof π for it, it can be easily checked (in $\mathcal{P}$) if indeed π proves T in the system. Note that here efficiency of the verification procedure refers to its running-time measured in terms of the *total length of the alleged theorem and proof.*

Remark 6.2. Note that I dropped the requirement used in the definition of $\mathcal{NP}$, limiting the proof to be short (polynomial in the length of the claim). The reason is, of course, that proof length is now our measure of complexity.

All these conditions are concisely captured, for propositional statements, by the following definition.

Definition 6.3 (Proof systems [CR79]). A (*propositional*) *proof system* is a polynomial-time algorithm M with the property that T is a tautology if and only if there exists a (*proof*) π such that $M(\pi, T) = 1$.

As a simple example, consider the following truth-table proof system M_{TT}. Basically, this machine will declare a formula T a theorem if evaluating it on every possible input makes T true. A bit more formally, for any formula T on n variables, the machine M_{TT} accepts (π, T) if π is a list of *all* binary strings of length n, and for each such string σ, $T(\sigma) = 1$.

Note that M_{TT} is indeed a proof system: It is sound, complete, and runs in polynomial time in its input length, which is the combined length of formula and proof. But in the system M_{TT}, proofs are of exponential length in the size of the number of variables, and so typically also exponential in the size of the given formula. This length is what we will care about. It leads us to the definition of the efficiency (or complexity) of a general propositional proof system M—how short the shortest proof of each tautology is.

Definition 6.4 (Proof length [CR79]). For each tautology T, let $\mathsf{S}_M(T)$ denote the size of the shortest proof of T in M (i.e., the length of the shortest string π such that M accepts (π, T)). Let $\mathsf{S}_M(n)$ denote the maximum of $\mathsf{S}_M(T)$ over all tautologies T of length n. Finally, we call the proof system M *polynomially bounded* iff for all n, we have $\mathsf{S}_M(n) = n^{O(1)}$.

---

[6]Actually, even the first two requirements are too much to expect from strong proof systems, as Gödel famously proved in his Incompleteness theorem. However, for propositional statements that have finite proofs, there are such systems.

Is there a polynomially bounded proof system (namely, one that has polynomial-size proofs for all tautologies)? The following theorem provides a basic connection of this question with computational complexity and the major question of Section 3.5. Its proof follows quite straightforwardly from the $\mathcal{NP}$-completeness of *SAT*, the problem of satisfying propositional formulas, the fact that a formula is unsatisfiable iff its negation is a tautology, and the observation that a short proof in any propositional proof system certifies (in the sense of $\mathcal{NP}$) such unsatisfiability.

Theorem 6.5 [CR79]. *There exists a polynomially bounded proof system if and only if $\mathcal{NP} = \text{co}\mathcal{NP}$.*

In the next section, we focus on natural restricted proof systems. Note that a notion of reduction between proof systems, called *polynomial simulation*, was introduced by Cook and Reckhow [CR79] and allows us to create a partial order on the relative power of some systems. This is but one example of the usefulness of the computational complexity methodology developed in complexity theory after the success of $\mathcal{NP}$-completeness.

6.3   Concrete proof systems

Almost all proof systems in this section are of the familiar variety, going back to the deductive system introduced in *The Elements* of Euclid for plane geometry. Each proof starts with a list of formulas (assumed "true"), and by a successive use of simple (and sound!) derivation rules, infers new formulas (each formula is called a *line* in the proof). Typically the starting formulas will be *axioms*, and the final derived formulas will be the proven theorem.

Here, and in this research area in general, it is often more convenient to focus on the related notion of *contradiction* systems, which prove a theorem by refuting its negation. More precisely, one starts with a contradictory set of formulas and derives a *basic*, patently evident contradiction (e.g. $\neg x \wedge x$, $1 = 0$, $1 < 0$), depending on the setting. We highlight some results and open problems on the proof length of basic tautologies in algebraic, geometric, and logical systems. In each of these sections, I give an example of a proof system in action, refuting the small contradiction ϕ in Figure 13, which has the five (mutually contradictory) "axioms" in four variables x, y, z, w.

$$A1 \quad \neg x \vee w$$
$$A2 \quad \neg w \vee y$$
$$A3 \quad \neg y$$
$$A4 \quad x \vee y \vee z$$
$$A5 \quad \neg z \vee x$$

Figure 13. The contradiction ϕ.

6.3.1   Algebraic proof systems

Here, a natural representation of a Boolean contradiction is a set of multivariate polynomials with no common root. Fix a field $\mathbb{F}$ (many, but not all, results hold for any field). To ensure that we consider only roots with Boolean $(0, 1)$ values, we always add to such a collection the polynomials $x^2 - x$ (for every variable x). These added "axioms" ensure that all possible roots are in $\mathbb{F}$ itself (no need for the algebraic closure). This also effectively makes all polynomials in the proof multilinear and greatly simplifies some of the issues that arise in general—for example, degrees of polynomials will never exceed the number of variables.

Note that it is always easy to encode a Boolean formula as a polynomial. Here is one way to represent the constraints of the pigeonhole principle PHP_n^m, defined above (Definition 6.1) as a set of contradictory (constant degree) polynomials. For every pigeon i, add the polynomial $\sum_j x_{ij} - 1$, and for every two pigeons i, i', and every hole j, add the polynomial $x_{ij}x_{i'j}$.

Now, let us discuss two concrete algebraic proof systems: Nullstellensatz (NS) and polynomial calculus (PC).

The Nullstellensatz (NS) proof system   The paper [BKI⁺96] suggests a proof system, based on Hilbert's Nullstellensatz, which states that if the polynomials $f_1, f_2, \ldots, f_m$ over n variables have no common root, then the constant function 1 is in the ideal generated by these polynomials. In other words, there must exist polynomials $g_1, g_2, \ldots, g_m$ such that

$$\sum_i f_i g_i \equiv 1.$$

Note that the coefficients g_i indeed constitute a proof: if the f_i's had a common root, such an identity could not hold. This ensures soundness. Completeness is given in Hilbert's famous Nullstellensatz, which in this Boolean setting is much easier to prove.

A natural measure of proof *length* is the description length of the polynomials as lists of all their coefficients (this is called a *dense* representation). Another important complexity parameter of Nullstellensatz proofs is *degree*: A proof has degree d if for all i, the polynomial $f_i g_i$ has degree at most d. Note that if d is the minimal degree of any proof, then the length of any proof in this dense representation is at least $\binom{n}{d}$ and at most $m(2n)^d$. Thus the degree d practically determines the proof length in this representation, and we shall thus focus on degree. Finally, note that proof verification in this system is easy; a simple polynomial-time algorithm can efficiently test if given g_i's satisfy $\sum_i f_i g_i \equiv 1$.

The polynomial calculus (PC) proof system   A related proof system, intuitively based on computations of Gröbner bases, is polynomial calculus, abbreviated PC, which was introduced by Clegg, Edmonds, and Impagliazzo [CEI96]. The *lines* in this system are polynomials (again represented explicitly by all coefficients), and it has two *deduction rules*, capturing the definition of an *ideal*: addition of two ideal elements, and multiplication of an ideal element by any polynomial. Specifically, for any two polynomials g, h and variable x_i, we can do the following:

- **Addition:** use g and h to derive $g + h$.

- **Extension:** use g to derive gx_i (or, more generally, derive gh).

Observe the *soundness* of the rules: if the input polynomials to the rule evaluate to 0 on a given variable assignment, so does the output polynomial. The task, as in Nullstellensatz, is to derive the constant 1 from the axioms. One may view such a proof as constructing the contradictory identity (as in Nullstellensatz) but in "small steps", which may yield smaller degree and description lengths of the polynomials appearing in the proof. Figure 14 describes a refutation, expanded as a tree, of the contradiction ϕ of Figure 13 in the system PC. Note that the axioms A1–A5 are encoded as polynomials and both deduction rules are used.

Automatizability   It is possible to show that for both proof systems above, if there is a proof of size s for some tautology, then this proof can be *found* in time polynomial in s. Indeed, as we are measuring size in the dense representation of polynomials, finding the proof in the Nullstellensatz system reduces to solving a linear system of size poly(s), whose variables are the coefficients of all polynomials appearing in the proof. For polynomial calculus, the proof finding algorithm given by [CEI96] is a simple variant of the Gröbner basis algorithm, which in our propositional setting requires polynomial time, as for the proof size as well.

A proof system with this property (namely, that short proofs, if they exist, may be efficiently found) is called *automatizable*, as one can efficiently automate proof search. Recalling our discussion on $\mathcal{P}$ vs. $\mathcal{NP}$ vs. co$\mathcal{NP}$ above, we do not expect that really strong propositional proof systems are automatizable.

The two systems above already illustrate that some types of reasoning can be more efficient than others. The PC system is known to be exponentially stronger than Nullstellensatz. More precisely, [CEI96] proves that there are tautologies requiring exponential-length Nullstellensatz proofs, but only polynomial-length PC proofs. However, strong size lower bounds, (obtained from degree lower bounds, as explained above) are known for the PC system as well. Indeed, the pigeonhole principle is hard for this system. For its natural
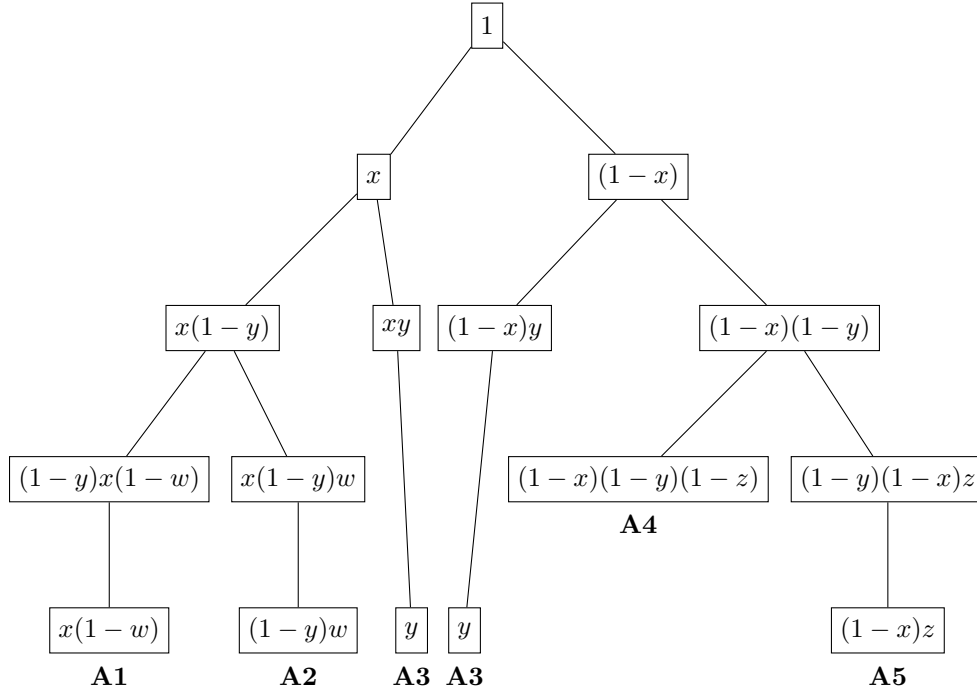
Figure 14. A tree-like polynomial calculus refutation of ϕ.

encoding of *PHP* as a contradictory set of quadratic polynomials as above, Razborov [Raz98b] proved the following.

Theorem 6.6 [Raz98b]. *For every n and every $m > n$, $\mathsf{S}_{\mathrm{PC}}(PHP_n^m) \geq 2^{n/2}$ over every field.*

6.3.2   Geometric proof systems

Yet another natural way to represent Boolean contradictions is by a set of regions in Real space containing no integer points. Rich sources of interesting contradictions are integer programs from combinatorial optimization. Here the constraints are (affine) linear inequalities with integer coefficients (so the regions are subsets of the Boolean cube carved out by halfspaces). More generally, certain relaxations of integer programs are expressed by systems of polynomial inequalities. A proof system infers new inequalities from old ones in a way that does not eliminate integer points.

We again demonstrate such geometric intuition with two concrete systems: cutting planes (CP) and sum-of-squares (SOS).

Cutting planes (CP) proofs   The most geometric proof system is called cutting planes (CP), introduced by Chvátal [Chv73]. Its *lines* are linear inequalities with integer coefficients. Its *deduction rules* are addition and integer division. Specifically, assume ℓ_i, m_i, a, b, c are integers.

- **Addition:** use $\sum \ell_i x_i \geq a$ and $\sum m_i x_i \geq b$ to infer $\sum (\ell_i + m_i) x_i \geq a + b$.

- **Integer division:** If c divides all m_i, use $\sum m_i x_i \geq b$ to infer $\sum (m_i/c) x_i \geq \lceil b/c \rceil$.

A refutation derives a basic contradiction (e.g., $0 \geq 1$) from the axioms. Figure 15 describes a refutation, expanded as a tree, of the contradiction ϕ of Figure 13, in the system CP. Note that the axioms A1–A5 are encoded as linear inequalities, and both deduction rules are used.

$$0 \geq 1$$

$$-x \geq 0 \qquad x \geq 1$$

$$y - x \geq 0 \qquad -y \geq 0 \qquad 2x \geq 1$$

$$w - x \geq 0 \qquad y - w \geq 0 \qquad -y \geq 0 \qquad 2x + y \geq 1$$

$$w + (1-x) \geq 1 \qquad (1-w) + y \geq 1 \qquad (1-y) \geq 1 \qquad x + y + z \geq 1 \qquad x - z \geq 0$$

A1   **A2**   **A3**   **A4**

$$(1-y) \geq 1 \qquad\qquad x + (1-z) \geq 1$$

A3   **A5**
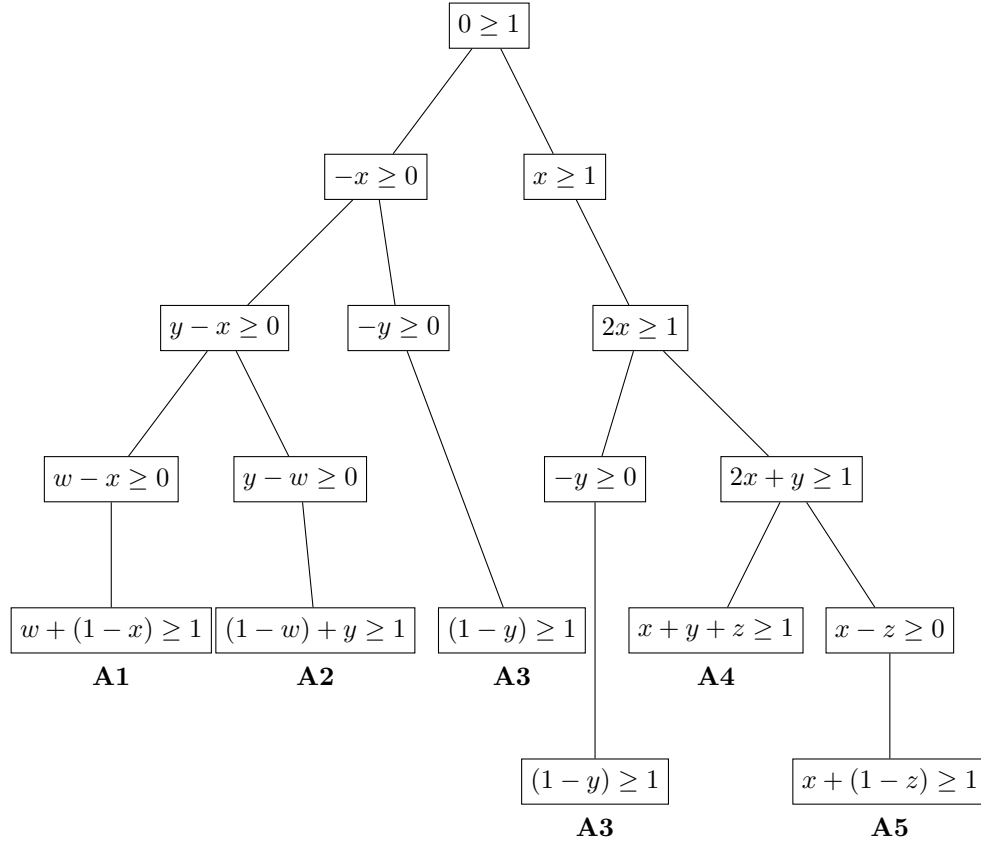
Figure 15. A tree-like cutting planes refutation of ϕ.

It is not hard to see that, if the original Boolean axioms are disjunctions (as in the contradiction ϕ), then when we translate them to linear inequalities as above, when they have a satisfying integer assignment, they also have a Boolean one. In other words, cutting planes is a sound and complete propositional proof system.

Consider again the pigeonhole principle PHP_n^m. First, let us express it as a set of contradictory linear inequalities: For every pigeon, the sum of its variables should be *at least* 1. For every hole, the sum of its variables should be *at most* 1. Thus, adding up all variables in these two ways implies $m \leq n$, a contradiction. Therefore, the pigeonhole principle has polynomial-size CP proofs.

While PHP_n^m is easy in this system, exponential lower bounds have been proved for other tautologies, and next I explain how. Consider the tautology $CLIQUE_n^k$: No graph on n nodes can simultaneously have a k-clique and a legal $(k-1)$-coloring. It is easy to formulate this tautology as a propositional formula. Notice that it somehow encodes many instances of the pigeonhole principle, one for every k-subset of the vertices.

Theorem 6.7 [Pud97]. $\mathsf{S}_{\mathsf{CP}}\big(CLIQUE_n^{\sqrt{n}}\big) \geq 2^{n^{1/10}}$.

The proof of this theorem by Pudlak [Pud97] is quite remarkable. It *reduces* this proof complexity lower bound to a circuit complexity lower bound. In other words, Pudlak shows that any short CP-proof of tautologies of certain structure yields a small circuit computing a related Boolean function (this is a general method, which is discussed in Section 6.4). You probably guessed that for the tautology at hand, the hard function to be used is indeed the *CLIQUE* function introduced earlier. Thus, if the resulting circuit is a *monotone* Boolean circuit (as in Section 5.2.3), we would be done by Theorem 5.14. As it turns out, the circuits Pudlak obtains are *monotone*, but they are stronger, as they are allowed to use real rather than Boolean values. More precisely, rather than having only $\wedge, \vee$ as basic gates, these circuits can use *any* monotone binary operation on real numbers as a gate: Their inputs and output must be

Boolean, but intermediate values can be arbitrary reals. Such circuits are indeed far stronger—they can solve *some* $\mathcal{NP}$-complete problems in linear size [Ros97]! Despite that, Pudlak proceeds to generalize Razborov's approximation method (Section 5.2.3) for such circuits and proves that even they require an exponential size to compute *CLIQUE*. A different proof, obtained independently by Haken and Cook, appears in [HC99]. An earlier, lower bound on tree-like CP-proof size, via a different method, is described in Section 15.2.4.

Sum-of-Squares (SOS) proofs   A much stronger geometric proof system (for polynomials over the reals) has recently emerged as important for optimization, machine learning, and complexity, called the sum-of-squares (SOS) system.[7] It was introduced in several papers [Sho88, Nes00, Par00, GV01, Las01], with motivations from optimization, statistics (moment problems), and proof complexity. Curiously, the origins of SOS can be traced back to Hilbert's seventeenth problem, which inspires this proof system in the same way his Nullstellensatz theorem inspired proof systems in the previous section. Recall that Hilbert's seventeenth problem concerns multivariate polynomials over the real numbers. It starts with the observation that every sum of squares of polynomials is everywhere nonnegative, and asks if the converse is true, namely, if *every* polynomial that is nonnegative everywhere can be written as a sum of squares (of rational functions). Hilbert's seventeenth was solved affirmatively by Artin [Art27]. Further development of these ideas led to the Positivstellensatz theorem of Krivine [Kri64], Stengle [Ste74], Putinar [Put93], and others, a cornerstone of real algebraic geometry, which gives a characterization of when a set of polynomial equations and inequalities has *no* common solution.

The SOS system (which we explain for simplicity only for refuting systems of equations), utilizes this characterization. It proves that a set of real polynomials $f_1, f_2, \ldots, f_n$ (with any number of variables) has no common root by exhibiting polynomials $g_1, g_2, \ldots, g_n$ and $h_1, h_2, \ldots, h_k$ such that

$$\sum_i f_i g_i \equiv 1 + \sum_j h_j^2.$$

Such a proof is said to have degree d if the degrees of all $f_i g_i$ and h_j^2 do not exceed d. Clearly, the SOS system is at least as strong as the Nullstellensatz system discussed in the previous section, in which squares cannot be used. Moreover, Grigoriev [Gri01a] gives examples showing that the SOS system can be exponentially stronger (i.e., tautologies),[8] for which Nullstellensatz or PC refutations require linear degree, but SOS proves them with constant degree (and thus in polynomial size). It turns out that SOS is also more powerful than the cutting-planes system CP, as well as several important linear and semidefinite based proof systems like those in [SA90, LS91].

Another property SOS shares with Nulstellensatz and PC is being *automatizable*.[9] Namely, if a system of polynomial equations (and inequalities) has a degree-d SOS proof, then this proof (which has size at most n^d, the number of coefficients needed) can actually be *found* in time $n^{O(d)}$ (using semidefinite programming). For constant d, such proofs (if they exist) can be found in polynomial time. This property is important for obtaining efficient approximation algorithms for the large variety of (nonconvex) problems that can be formulated as follows: Find the maximum value that a given multivariate polynomial attains in a semi-algebraic subset of $\mathbb{R}^n$, one defined by polynomial equations and inequalities (say, all of constant degree). SOS provides a hierarchy of approximation algorithms for this optimal value that are parameterized by the degree d of polynomials used in proving this approximation. As d increases, the quality of the approximation typically improves, while the running time increases. Numerous applications of this algorithm are known (see, e.g., Lasserre [Las09]). Recent applications and connections to complexity theory, machine learning, quantum information, and more are surveyed by Barak and Steurer [BS14]. Crucial to many of these applications is the fact that many basic inequalities (Cauchy-Schwarz, Hölder, hypercontractive inequalities for polynomials, triangle inequalities for various norms, etc.) have *constant-degree* SOS-proofs.[10]

---

[7]It is also referred to in the literature as Positivstellensatz or Lasserre.

[8]For example, that $x_1 + x_2 + \cdots + x_n = \frac{1}{2}$ has no 0/1-solution.

[9]Although one should consider the caveats and precise statement in [O'D17].

[10]For example, here is a degree-4 proof of the Cauchy-Schwarz inequality:
$(\sum_i^n x_i^2)(\sum_i^n y_i^2) - (\sum_i^n x_i y_i)^2 = \frac{1}{2}(\sum_{i \neq j}(x_i y_j - x_j y_i)^2) \geq 0.$

Which tautologies are hard for this SOS system? As usual, we are mainly interested here in discrete problems (i.e., polynomial equations over Boolean variables) as in the previous section. Their encoding as real polynomials is easily achieved in the same way, by adding the polynomials $x_i^2 - x_i$ as axioms. In this setting, (as polynomials are multilinear without loss of generality), it is not hard to see that proofs never require degree larger than n. One of the strongest results we have is a linear degree lower bound for almost all inconsistent systems of linear equations over $\mathbb{F}_2$.

Theorem 6.8 [Gri01b,Sch08]. *For every n, let $f_1, f_2, \ldots, f_{10n}$ be randomly and independently chosen linear equations over n variables of the form $x_i + x_j + x_k = b$ (where i, j, k are uniformly random in $[n]$ and b is random in $\{0, 1\}$). Then with probability $1 - o(1)$, the encoded system of real polynomials has no common root, and every SOS refutation requires degree $\Omega(n)$.*

The relation between degree and size of SOS proofs (namely, that degree d proofs can be found in time $n^{O(d)}$ via semidefinite programming, and hence have size at most $n^{O(d)}$) is shown to be tight, for example, for $4 - SAT$ in [LN15]. A tight relation between size of semidefinite programs and degree of SOS proofs, implying exponential SDP lower bounds, is established in the breakthrough by Lee, Raghavendra, and Steurer [LRS14].

6.3.3   Logical proof systems

The proof systems in this section will all have *lines* that are Boolean formulas, and the differences between these systems will be in the structural limits imposed on these formulas. We introduce the most important ones: Frege, capturing "polynomial time reasoning," and Resolution, the most useful system used in automated theorem provers.

The Frege proof system   The most basic proof system, called the "Frege system," puts no restrictions on the formulas manipulated by the proof. As a refutation system, it has one nontrivial *derivation rule*, called the *cut rule* (or Modus Ponens):

- **Cut rule:** Use formulas $A \vee C$, $B \vee \neg C$ to infer the formula $A \vee B$.

Other derivation rules allow, for example, taking the conjunction of two previously derived formulas, as well as taking the disjunction of a previously derived formula with an arbitrary one. As usual, a refutation should derive a contradiction (e.g., the empty clause, or $x \wedge \neg x$), from the given axioms. The *size* of a Frege proof is simply the total size of all formulas appearing in it.

Every basic book in logic has a slightly different way of describing the Frege system. One convenient outcome of the computational approach, especially the notion of efficient reductions between proof systems, is a proof (by Cook and Reckhow [CR79]) that they are *all* equivalent, in the sense that the shortest proofs (up to polynomial factors) are independent of which variant you pick.

The Frege system can polynomially simulate *both* the Polynomial Calculus[11] and the Cutting Planes systems. In particular, the "counting" CP proof described above for the pigeonhole principle can be carried out efficiently in the Frege system (not quite trivially!), yielding the following theorem.

Theorem 6.9 [Bus87]. (PHP_n^{n+1}) *has* Frege *proofs of size* $n^{O(1)}$.

Frege systems are basic in the sense that they are the most common in logic and in that polynomial-length proofs in these systems naturally correspond to "polynomial-time reasoning" about feasible objects. In a sense, Frege is the proof-complexity analog of the computational class $\mathcal{P}$.[12] The major open problem in proof complexity is to find any tautology (as usual, I mean a family of tautologies) that has no polynomial-size proof in the Frege system.

Open Problem 6.10. Prove super-polynomial lower bounds for the Frege system.

---

[11]This is simple over the binary field, and with appropriate representation applies to other fields as well.

[12]A variant of Frege, called Extended-Frege, operates with circuits instead of formulas as lines in the proof (with similar derivation rules) and may perhaps better capture polynomial time reasoning.

The resolution proof system   As lower bounds for Frege are hard, we turn to subsystems of Frege that are interesting and natural. The most widely studied system is Resolution. Its importance stems from its use by most propositional (as well as first-order) *automated theorem provers*, often called "Davis–Putnam" or "DLL" procedures [DLL62]. This family of algorithms is designed to find proofs of Boolean tautologies arising in diverse computer science applications, from verification of software and hardware designs and communication protocols to automatic generation of proofs of basic number theory and combinatorial theorems.

The *lines* in Resolution refutations are *clauses*, namely, disjunctions of literals (like $x_1 \lor x_2 \lor \neg x_3$). The *cut rule* in Frege simplifies here to the *resolution rule*:

- **Resolution rule:** Use clauses $A \lor x$ and $B \lor \neg x$ to derive the clause $A \lor B$.

A Resolution refutation starts with a set of mutually unsatisfiable clauses (axioms) and derives the empty clause (a contradiction) via repeated application of the resolution rule above. The *size* of a Resolution proof may be taken simply as the number of clauses in the proof (as no disjunction has size larger than n). Note that Resolution is the restriction of Frege in which one is only allowed to use the simplest type of formulas—namely, clauses—as lines in the proof.

Figure 16 describes a refutation, expanded as a tree, of the contradiction ϕ of Figure 13, in the Resolution proof system.
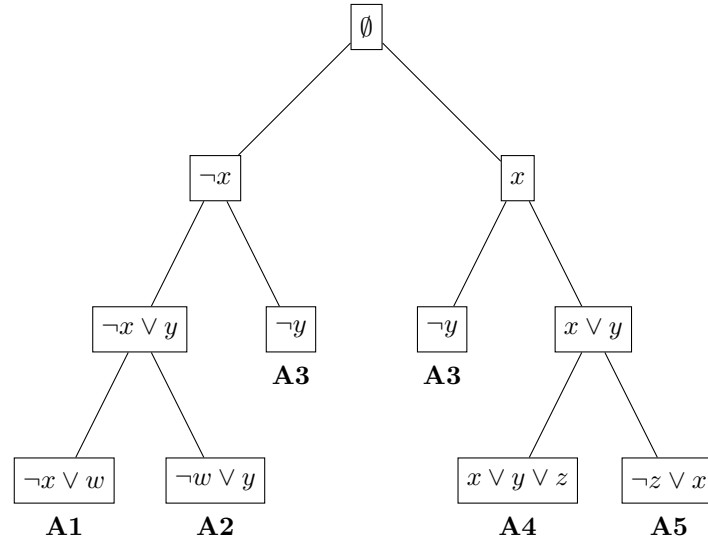
Figure 16. A tree-like Resolution refutation of ϕ.

Historically, the first major result of proof complexity was Haken's[13] [Hak85] exponential lower bound on Resolution proofs for the pigeonhole principle.

Theorem 6.11 [Hak85]. (PHP_n^{n+1}) *requires* Resolution *proofs of size* $2^{\Omega(n)}$.

To prove this theorem, Haken developed the *bottleneck method*, which is related to both the random restriction method and approximation method mentioned in the circuit complexity Section 5.2. This lower bound was extended by Chvátal and Szemerédi to *random tautologies* in [CS88]. A bit more precisely, they proved that picking sufficiently many clauses at random not only renders them mutually unsatisfiable with high probability, but also demonstrating this unsatisfiability will almost surely require exponentially large Resolution proofs. The *width method* developed by Ben-Sasson and Wigderson in [BSW99] unifies and provides much simpler proofs of these and other results. Moreover, it uncovers the role of graph *expansion* (discussed in Section 8.7) in many proof complexity lower bounds.

---

[13]Armin Haken, the son of Wolfgang Haken, cited earlier for his work on knots and the 4-color theorem.

The question of efficiently finding short Resolution proofs when they exist (in other words, how automatizable is this system?) is extremely interesting due to its prevalent use for automated theorem proving. The best known bounds (respectively in [BKPS02, AR08]) are that size s Resolution proofs for tautologies on n variables can be found in time $\exp(\sqrt{n \log s})$, but under a natural complexity assumption must take time at least $s^{\log n}$. Can any of them be improved to narrow this wide gap?

6.4   Proof complexity vs. circuit complexity

These two areas look like very different beasts, despite the syntactic similarity between the local evolution of computation and proof. To begin with, the number of objects they care about differs drastically. There are doubly exponentially many functions (on n bits), but only exponentially many tautologies of length n. Thus, a counting argument shows that some functions (albeit nonexplicit) require exponential circuit lower bounds (Theorem 5.6), but no similar argument can exist to show that some tautologies require exponential size proofs. So while we prefer proof-length lower bounds for natural, explicit tautologies, even nonconstructive *existence* results of hard tautologies for strong proof systems are interesting in this setting.

Despite the different natures of the two areas, there are deep connections between them. Quite a few of the techniques used in circuit complexity, most notably *random restrictions*, were useful for proof complexity as well. The lower bound of Pudlak on Cutting Planes, which we saw in Theorem 6.7, uses circuit lower bounds in an extremely intriguing way: A monotone circuit lower bound directly implies a (nonmonotone) proof system lower bound. This particular type of reduction is known as the "feasible interpolation method" (it may be viewed as a quantitative version of *Craig's interpolation* from first-order logic), which we define next.

Feasible interpolation   A proof system has *feasible interpolation* if whenever it proves in size s that a statement of the form $F(x, y) \vee G(x, z)$ (in disjoint sets of variables x, y, z) is a tautology, then there is a Boolean circuit of size poly(s) on inputs x, which identifies whether F or G is a tautology when these x variables are so fixed (of course, at least one of them must be, and it is possible that both are, in which case any output is good).

The feasible interpolation method was introduced by Krajíček [Kra94] and (more implicitly) by Razborov [Raz95b] for proving Resolution lower bounds. They noticed that for appropriate tautologies, the small circuit guaranteed by feasible interpolation can be made *monotone*. Hence monotone lower bounds on circuits can be used for Resolution lower bounds. This method was first used for Cutting Planes by Bonet, Pitassi, and Raz [BPR97], and is known for other relatively weak proof systems. We note that feasible interpolation is a weaker property than *automatizability* discussed above, and so the algebraic systems NS and PC also have it. However, the reader should check that if feasible interpolation holds for *every* propositional proof system, then $\mathcal{NP} \cap \text{co}\mathcal{NP} \subseteq \mathcal{P}/\text{poly}$ (and so we do not expect it of strong proof systems). Indeed, Krajíček and Pudlak [KP89] show that if feasible interpolation holds for the standard Frege system, then integer factoring is easy (and more generally, one-way functions (see Section 4.5) do not exist).

This connection raises the question of whether one can use reductions of a similar nature to obtain lower bounds for strong systems (like Frege) from (yet unproven) circuit lower bounds.

Open Problem 6.12. Does $\mathcal{NP} \not\subseteq \mathcal{P}/\text{poly}$ imply super-polynomial Frege lower bounds?

Why are Frege lower bounds hard? The truth is, we do not know. The Frege system (and its relative, Extended Frege), capture *polynomial-time reasoning*, because the basic objects appearing in the proof are polynomial-time computable. Thus, super-polynomial lower bounds for these systems are the proof complexity analog of proving super-polynomial lower bounds in circuit complexity. As we saw for circuits, we at least understand to some extent the limits of existing techniques, via natural proofs. However, there is no known analog of this framework for proof complexity.

I conclude with a tautology capturing the $\mathcal{P}/\text{poly}$ vs. $\mathcal{NP}$ question. The proof complexity of this tautology may further illuminate why proving circuit lower bounds is difficult.

This tautology, suggested by Razborov [Raz95a, Raz96], simply encodes propositionally the statement $\mathcal{NP} \not\subseteq \mathcal{P}/\text{poly}$, namely, that *SAT* does not have small circuits. More precisely, fix n, an input size to *SAT*,

and s, the circuit size lower bound we attempt to prove.[14] The variables of our "lower bound" formula LB_n^s encode a circuit C of size s. The formula LB_n^s simply checks that the function computed by C disagrees with SAT on at least one instance ϕ of length n. Namely, that either $\phi \in SAT$ and $C(\phi) = 0$ or $\phi \notin SAT$ and $C(\phi) = 1$. Note that the description of this tautology LB_n^s has size $N = 2^{O(n)}$, so we seek a super-polynomial in N lower bound on its proof length.[15]

Proving that LB_n^s is hard for Frege will in some sense give another explanation of the difficulty of proving circuit lower bounds. Such a result would be analogous to the one provided by natural proofs, only without relying on the existence of *one-way* functions. But paradoxically, the same inability to prove circuit lower bounds seems to prevent us from proving this proof complexity lower bound!

Even proving that LB_n^s is hard for Resolution has been extremely difficult. It involves proving hardness of a *weak* pigeonhole principle[16]—one with exponentially more pigeons than holes. After several partial results, this was achieved by the tour-de-force of Raz [Raz04a]; and the further strengthening by Razborov [Raz04b] (for the so-called functional, onto pigeonhole principle) finally implies the hardness of LB_n^s for Resolution.

---

[14]For example, we may choose $s = n^{\log \log n}$ for a super-polynomial bound, or $s = 2^{n/1000}$ for an exponential one.

[15]Of course, if $\mathcal{NP} \subseteq \mathcal{P}/\text{poly}$, then this formula is *not* a tautology, and there is no proof at all.

[16]This explains the connection mentioned between the pigeonhole principle and the counting argument proving existence of hard functions.

7 Randomness in computation

The marriage of randomness and computation has been one of the most fertile ideas in computer science, with a wide variety of models ranging from cryptography to computational learning theory to distributed computing. It enabled new understanding of fundamental concepts, such as knowledge, secrecy, learning, proof, and indeed, randomness itself. In this and the next chapter, we shall just touch the tip of the iceberg— things most closely related to the questions of efficient computation and proofs. Sections 7.1 and 7.2 tell the (seemingly) contradictory stories of the power and weakness of algorithmic randomness. Section 7.3 explains the notion of pseudo-randomness, and pseudo-random generators, explaining why we tend to believe that the power of randomness in algorithms is not as strong as it might seem. Good sources for more information are [MR95], [Gol99] and [Vad11].

7.1 The power of randomness in algorithms

Let us start with an example that illustrates a potential dilemma encountered by mathematicians who try to prove identities. Assume we are working over the rationals $\mathbb{Q}$. The $n \times n$ Vandermonde matrix $V(x_1, \ldots, x_n)$ in n variables has $(x_i)^{j-1}$ in the (i, j) position. The Vandermonde identity is the following.

Proposition 7.1. $\det V(x_1, \ldots, x_n) \equiv \prod_{i<j}(x_i - x_j)$.

While this particular identity is simple to prove, many others like it are far harder. Suppose you conjectured an identity $q(x_1, \ldots, x_n) \equiv 0$, concisely expressed (as above) by a short formula, and wanted to know whether it is true before investing much effort in proving it.[1] Of course, if the number of variables n and the degree d of the polynomial q are large (as in the example), expanding the formula to check that all coefficients vanish will take exponential time and is thus infeasible. Indeed, no subexponential time algorithm for this problem is known! Is there a quick and dirty way to find out?

A natural idea suggests itself: Assuming q is *not* identically zero, then the algebraic variety it defines (the points at which q vanishes) has measure zero. So, if we pick *at random* values for the variables, chances are we shall miss this variety, and hit a nonzero of q. If however q is identically zero, then *every* assignment we pick will evaluate to zero. It turns out that the random choices can actually be restricted to a finite domain, and the following can be simply proved by induction on n.

Proposition 7.2 [DL78, Zip79, Sch80]. *Let q be a* nonzero *polynomial of degree at most d in n variables. Let r_i be uniformly and independently chosen from*[2] $\{1, 2, \ldots, 3d\}$. *Then* $\Pr[q(r_1, \ldots, r_n) = 0] \leq 1/3$.

Note that since evaluating the polynomial q at any given point is easy given a formula for f, the above constitutes an efficient *probabilistic* algorithm for verifying polynomial identities. Probabilistic algorithms differ from the algorithms we have seen so far in two ways. First, they are able to toss independent, unbiased coins and use the outcomes in the computation. Thus, the output of a probabilistic algorithm is a random variable. Second, probabilistic algorithms make errors. The beauty is that if we are willing to accept both the availability of perfect randomness as extra input, and the presence of small errors in the output, we seem to be getting far more efficient algorithms for seemingly hard problems.

The deep issue of whether randomness exists in nature[3] has never stopped humans from assuming it anyway, for gambling, tie breaking, polls, and more. Perhaps Nature provides some randomness (such as sun spots, radioactive decay, weather, stock-market fluctuations, or Internet traffic), but actual physical measurements of these unpredictable events do not produce perfectly independent and unbiased coin tosses. The question of whether such *weak sources* of randomness can be used in probabilistic algorithms, and the

---

[1]This problem turns out to be even more fundamental than it may seem here. It is called the *Polynomial Identity Testing* problem (or PIT for short) and is deeply related to arithmetic complexity theory, the subject of Chapter 12. This problem is discussed at length, for example, in section 4 of [SY10].

[2]A general principle used here and throughout is that the access to random independent *bits* gives easy access to (essentially) uniform random samples from any finite range.

[3]What quantum mechanics says about it will be discussed in Chapter 11.

theory developed for it, will be discussed in Chapter 9. *Here we postulate access of our algorithms to perfect coin flips, and develop the theory from this assumption.*[4]

The presence of error in probabilistic algorithms seems like a serious issue—after all, we compute to discover a *fact*, not a "maybe." However, we do tolerate uncertainty in real life (not to mention computer hardware and software errors) anyway, so it makes sense to allow it in algorithms as well. Moreover, observe that the error of a probabilistic algorithm is much more controllable than errors in other situations—here it can be decreased arbitrarily, with a small penalty in efficiency. Assume that our algorithm makes an error of at most $1/3$ on any input (as in the proposition above). For any algorithm with this property, running it k times (with independent random choices each time) and taking a majority vote of the answers would reduce the error to $\exp(-k)$ on every input.[5]

Thus we revise our notion of efficient computation to allow probabilistic algorithms with small error and define the probabilistic analog $\mathcal{BPP}$ (for bounded error, probabilistic, polynomial time) of the class $\mathcal{P}$. We note that one can (and does) define probabilistic analogs of other deterministic complexity classes and can study the power of randomness in these settings as well.

Definition 7.3 (The class $\mathcal{BPP}$ [Gil77])**.** The function $f\colon \mathbf{I} \to \mathbf{I}$ is in $\mathcal{BPP}$ if there exists a probabilistic polynomial-time algorithm A, such that for every input x, $\Pr[A(x) \neq f(x)] \leq 1/3$.

In this definition, $A(x)$ denotes the random variable that is the output of the probabilistic algorithm A. Sometimes it is notationally more convenient to make explicit mention of the random bits used in the algorithm, namely, to consider $A(x)$ as $A'(x, r)$. Here A' is a deterministic algorithm, which (besides the actual input x) receives an auxiliary input r (of appropriate length) that is assumed to be a uniformly distributed sequence of random bits. In this notation, the requirement in the definition can be written as $\Pr_r[A'(x, r) \neq f(x)] \leq 1/3$ for a deterministic algorithm A' that runs in polynomial time in $|x|$.

I stress again that the probability bound in this definition is over the internal coin tosses r of the algorithm and must hold for *every* input. This definition of $\mathcal{BPP}$ is extremely robust to changes of the error probability bound: Replacing $1/3$ by the lower $1/10^{10}$, or even $\exp(-|x|)$, as well as by the higher $.49999$ or even $1/2 - 1/\mathrm{poly}(|x|)$, leaves the definition unchanged (this follows by the error-reduction via majority idea described above).

Remark 7.4. $\mathcal{BPP}$ is a class of *decision* problems. Throughout this chapter we will also consider probabilistic algorithms for relations, approximation problems, counting problems and others. We will not formally define the complexity classes for them; intuitively, in all, we seek a probabilistic algorithm that will efficiently deliver, for every input, a desired output with high probability. Essentially everything stated regarding $\mathcal{BPP}$ holds for these classes as well.

Adleman [Adl78] observed that in probabilistic algorithms with such tiny errors, some (indeed, most) random strings are simultaneously good for every input of a given length. Allowing nonuniformity, any of them can be hard-wired into a *circuit* that will compute correctly on every input. Thus, for any problem in $\mathcal{BPP}$, there *exist* small circuits.[6]

Theorem 7.5 [Adl78]**.** $\mathcal{BPP} \subseteq \mathcal{P}/\mathrm{poly}$.

Probabilistic algorithms were used in statistics (for sampling) and physics (Monte Carlo methods — see Section 20.1.5) before computer science existed. However, their introduction into computer science, starting with the probabilistic polynomial factoring algorithms of Berlekamp [Ber67] and the probabilistic primality tests of Solovay and Strassen [SS77] and Rabin [Rab80], was followed by an avalanche of results that increased the variety and sophistication of problems amenable to such attacks tremendously—a glimpse into this scope

---

[4]In practical implementations of probabilistic algorithms, these bits are usually generated by a variety of ad hoc "pseudo-random generators." It is a remarkable empirical fact that almost universally, these ad hoc alternatives to random bits seem to work pretty well.

[5]The notation $\exp(-k)$ means c^{-k} for some $c > 1$. This bound on the error follows from standard concentration bounds on the binomial distribution (e.g., the Bernstein/Chernoff bound). Specifically, the probability that k tosses of a biased coin, whose probability of heads is at most $1/3$, would produce more than $k/2$ heads, is exponentially small in k.

[6]Recall that in Section 5.2, we denoted the class of polynomial-size circuits by $\mathcal{P}/\mathrm{poly}$.

can be obtained from Motwani and Raghavan's textbook [MR95]. We restrict ourselves here only to those which save *time*, and note that randomness seems to help save other resources as well.

I list here a variety problems from different areas that have probabilistic polynomial time algorithms,[7] but for which the best known deterministic algorithms require exponential time. These are among the important achievements of this research area.

- **Generating primes.** Given an integer x (in binary), produce a prime in the interval $[x, 2x]$ (note that this interval is exponentially long in the input length $|x|$). The prime number theorem guarantees that a random number in this interval is a prime with probability about $1/|x|$ (so one will show up in polynomial time in $|x|$ with high probability, and we can check its primality efficiently).

- **Polynomial factoring** (Kaltofen [Kal83]). Given an arithmetic formula or circuit[8] describing a multivariate polynomial (over a large finite field), find its irreducible factors.[9]

- **Permanent approximation** (Jerrum, Sinclair, and Vigoda [JSV04]). Given a nonnegative real matrix, approximate its permanent (defined in Chapter 12) to within (say) a factor of 2. Note that unlike its relative, the determinant, which can be easily computed efficiently by Gaussian elimination, the permanent is known to be $\#\mathcal{P}$-complete (which implies $\mathcal{NP}$-hardness) to compute exactly.

- **Volume approximation** (Dyer, Frieze, and Kannan [DFK91]). Given a convex body in high dimension (e.g., a polytope given by its bounding hyperplanes), approximate its volume to within (say) a factor of 2. Again, computing the volume exactly is $\#\mathcal{P}$-complete.

The most basic question about this new computational paradigm of probabilistic computation is whether it really adds any power to deterministic computation.

Open Problem 7.6. Is $\mathcal{BPP} = \mathcal{P}$?

The answer seemed to be negative: We had no idea in sight as to how to solve the above-mentioned problems, and many others, by a deterministic algorithm running even in subexponential time, let alone in polynomial time. However, the next section should radically change this viewpoint, through the fundamental notion of *computational pseudo-randomness*.

7.2   The weakness of randomness in algorithms

Let us start from the bottom line: If any of the numerous $\mathcal{NP}$-complete problems we saw above is *hard* then randomness is *weak*. There is a trade-off between what the words *hard* and *weak* formally mean. To be concrete, we give perhaps the most dramatic such result, due to Impagliazzo and Wigderson [IW97].

Theorem 7.7 [IW97]. *If SAT cannot be solved by circuits of size $2^{o(n)}$, then $\mathcal{BPP} = \mathcal{P}$. Moreover, the conclusion holds if SAT is replaced in this statement by any problem that cannot be solved by circuits of size $2^{o(n)}$ but has $2^{O(n)}$-time algorithm.*[10]

Rephrasing, exponential circuit lower bounds on essentially any problem of interest imply that randomness can *always* be eliminated from algorithms without sacrificing efficiency (up to a polynomial). Many variants of this result, which is generally called "*de-randomization*," exist. One variant gives a hardness-randomness trade-off: Weakening the assumed lower bound on the hard problem simply weakens the deterministic simulation of randomness but leaves it highly nontrivial. (In other words, the resulting deterministic algorithm substituting for the probabilistic one is far more efficient than brute force enumeration of the values for the

---

[7]Note that these problems are not strictly in $\mathcal{BPP}$, because they compute relations or approximation problems.

[8]See precise definitions in Chapter 12.

[9]It is not even clear that the output has a representation of polynomial length—but it does! A structural corollary of this result is that the factors have small arithmetic circuits as well.

[10]The class with such algorithms includes most $\mathcal{NP}$-complete problems but also presumably far more complex ones (e.g., determining optimal strategies of games, which are $\mathcal{PSPACE}$-complete, and beyond).

random bits). For example, if $\mathcal{NP} \not\subseteq \mathcal{P}/\text{poly}$, then $\mathcal{BPP}$ has deterministic algorithms with subexponential runtime $\exp(n^\varepsilon)$ for every $\varepsilon > 0$.

Another important extension is replacing the nonuniform circuit lower bound by a uniform hardness assumption (of the type $\mathcal{BPP} \neq \mathcal{NP}$). This results in an "average-case" de-randomization, as was defined and proved in Impagliazzo and Wigderson [IW98].

Note one remarkable and counterintuitive feature of such results: They assert that *if one computational task is hard, then another is easy!*

In light of Theorem 7.6, we are now faced with deciding which of two extremely appealing beliefs to drop (as they are contradictory). The first is that some natural problems (e.g., $\mathcal{NP}$-complete ones) cannot be solved efficiently. The second is that randomness is a very powerful algorithmic resource. Experience, intuition, and state-of-art knowledge seem to support both, but they seem far stronger in support of the first. So most experts reluctantly drop the second and now believe that randomness cannot significantly speed up algorithms. Namely, that probabilistic algorithms can be replaced by deterministic ones for the same task which are not much more costly—a statement that usually goes under the name *de-randomization*. We state it for classification problems (similar theorems for search and approximation problems are known as well, but we shall not discuss them here).

Conjecture 7.8. $\mathcal{BPP} = \mathcal{P}$.

We now construct a high-level description of the ideas leading to this surprising set of results, which are generally known under the heading *hardness vs. randomness*.[11] The central notions that make this *de-randomization* possible are *computational pseudo-randomness* and *pseudo-random generator*. I explain both concepts here, and we will see how they yield de-randomization. In Section 7.3, we will describe their history and importance, beyond de-randomization, and discuss different pseudo-random generators. We refer the reader to Chapter 8 of Goldreich's book [Gol08] and his monograph [Gol99] for more detail.

We are after a general way of eliminating the randomness used by any (efficient) probabilistic algorithm. Fix any such algorithm A. It has two kinds of inputs. The "real" input x, and the "randomness" y, which, let us say, is n bits long. The error guarantee is that for every x, if y is distributed according to the uniform distribution U_n on all binary sequences of length n, then $A(x, y)$ will err with probability at most $1/3$. The idea is to "fool" A, replacing the distribution U_n by another distribution D, which "looks like" U_n to A. Put differently, A will not be able to *distinguish* D from U_n on any input x. And more precisely, whether y is distributed according to U_n or to D, on every x, $A(x, y)$ will accept with nearly the same probability. Let us first define distributions that can "fool" *all* efficient algorithms. We will later see that to actually use such distributions, we need to make them *useful*, and discuss in which sense.

7.2.1 Computational pseudo-randomness

The first key notion we define is *computational pseudo-randomness*[12] of Goldwasser-Micali [GM84] and Yao [Yao82b]. For simplicity, we shall define it here with respect to circuits. For a circuit C with n inputs and a distribution D on n-bit sequences, denote by $C(D)$ the probability that $C(y) = 1$ when y is drawn from D. As usual, all definitions should be viewed asymptotically. In particular, when discussing a family of circuits $\mathcal{C}$, we will implicitly mean a sequence of circuits $\{\mathcal{C}_n\}$ parameterized by the input length n, with $\mathcal{C}_n$ typically restricted in size as a function of n. As in other places in this book, I will abuse notation and omit n (e.g., identifying $\mathcal{C}$ with $\mathcal{C}_n$ when the parameter n is implicit).[13] A schematic is shown in Figure 17.

Definition 7.9 (Pseudo-randomness). Let $\mathcal{C}$ be a family of circuits (on n bits), and $\epsilon > 0$. A distribution D (on n bits) is called $(\mathcal{C}, \epsilon)$-*pseudo-random* if for every $C \in \mathcal{C}$ we have $|C(D) - C(U_n)| \leq \epsilon$.

---

[11]The title of Silvio Micali's PhD thesis. Micali, along with his advisor Manuel Blum, constructed the first hardness-based pseudo-random bit generator.

[12]I often omit the "computational" and call it only "pseudo-randomness" in this chapter.

[13]Indeed, I may even change the number of inputs to be some polynomial in n without warning, when this does not affect the argument.
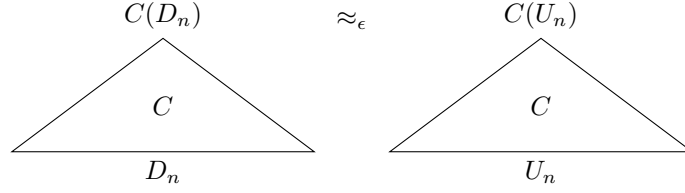
Figure 17. Schematic of a pseudo-random distribution D_n ϵ-fooling a circuit C.

In other words, D is pseudo-random if no circuit in $\mathcal{C}$ can "tell it apart" from the uniform distribution, with non-negligible advantage ϵ. Equivalently, D ϵ-fools $\mathcal{C}$.

The parameter ϵ can be a small constant (e.g., .01) for the purposes of this section, but in general, it may depend on n, with $\epsilon = \epsilon(n)$ tending to zero with n, often like $1/\text{poly}(n)$. The class of circuits can be arbitrary, but for this section, it is natural to take it to be $\mathcal{P}/\text{poly}$, or even better, the family of circuits of some fixed polynomial size, say, n^4. The reason is that for every efficient algorithm we are trying to de-randomize, if it runs in time (say) n^2, then a circuit can simulate its action on every n-bit input in time n^4 (as we saw in Section 5.2).

As we seek deterministic emulation of probabilistic algorithms, we must find a pseudo-random distribution D that can be efficiently generated from 0 random bits. But this is clearly impossible, even if we remove the efficiency requirement, as randomness cannot be generated deterministically.[14] Luckily, we have some leverage. Let us instead try to generate such pseudo-random D on n bits from *fewer* random bits, say, m. This leads us to the next important notion of a pseudo-random generator, stretching a few truly random bits into many pseudo-random ones, defined in the same paper of Yao [Yao82b].

7.2.2 Pseudo-random generators

Definition 7.10 (Pseudo-random generators). Let $\mathcal{C}$ be a family of circuits. A function $G : \{0,1\}^m \to \{0,1\}^n$ is called a $(\mathcal{C}, \epsilon)$-*pseudo-random generator* if, on uniformly random input, its output distribution $G(U_m)$ is $(\mathcal{C}, \epsilon)$-pseudo-random.

A schematic is shown in Figure 18. In this definition, we again think asymptotically, parameterizing everything by n, the length of the random input in circuits/algorithms we want to fool (which is also the output length of the generator). So G should be a family of functions $\{G_n : \{0,1\}^{m(n)} \to \{0,1\}^n\}$ that fools circuits from $\mathcal{C}_n$ with error $\epsilon(n)$.
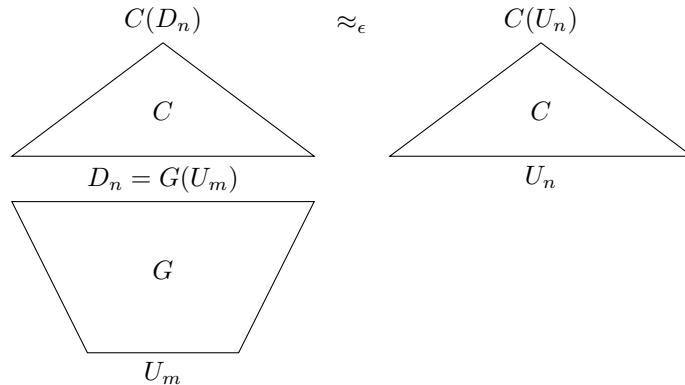
Figure 18. Schematic of a pseudo-random generator G ϵ-fooling a circuit C.

---

[14]The output of a deterministic process on a fixed input is, well, completely determined.

Pseudo-random generators can be used to fool uniform probabilistic algorithms, as their execution on each input can be simulated by circuits (see Section 5.2). For example, if we have such a pseudo-random generator G with $\mathcal{C}$ being the class of size n^4-size circuits, and with $\epsilon = \frac{1}{20}$ (say), then we can use $D = G(U_m)$ to fool any probabilistic n^2-time algorithm A on every input x for it. Because by definition, the error of the algorithm under this distribution would be at most $\frac{1}{3} + \frac{1}{20} < \frac{2}{5}$, which is crucially (as we shall presently see) bounded away from $\frac{1}{2}$.

But we don't have m random bits to generate D, because we aim for a deterministic algorithm. One simple, brute-force[15] way to get a *deterministic* emulation is to go over all possible sequences $z \in \{0,1\}^m$, compute $y = G(z)$ for each, and use the algorithm A with randomness y (namely, compute $A(x,y)$ for each such y). After getting all these 2^m results, we take their majority vote and declare it our answer for x. Note that this deterministic procedure will *always* give the correct answer for every input x (again, this is true since less than $\frac{1}{2}$ in the support of D causes an error, and we take a majority vote over the whole support).

The running time of this *de-randomized* algorithm depends on two factors: 2^m and the time complexity of G (indeed, on their product). Let us deal with them in turn. First, a fairly standard counting argument shows that there *exists* a function G with $m = O(\log n)$ that is a $(\mathcal{C}, \epsilon)$-pseudo-random generator. Indeed, a random G will have this property with very high probability. So, 2^m can be polynomial in n, which takes care of the first factor. However, a random G will typically be exponentially hard to compute. What we are looking for is a generator G that can be computed *efficiently*, in polynomial time. This would make the second factor polynomial as well and would yield a polynomial time emulation of $\mathcal{BPP}$. Are there such *efficient* pseudo-random generators? This is exactly what Theorem 7.7 provides, *assuming* that a sufficiently hard function exists.[16] This conversion of hardness into pseudo-randomness is also discussed in the next section.

7.3   Computational pseudo-randomness and pseudo-random generators

We now explore the history and significance of these two central notions. Section 7.2 presented them as extremely natural for the purpose of de-randomizing probabilistic algorithms. And indeed they are. But they were not born that way at all, but rather from cryptography and circuit complexity! This story demonstrates again the wonderful flow of ideas within computational complexity.

7.3.1   Computational indistinguishability and cryptography

The concept of computational pseudo-randomness is actually an (important) special case of a much broader notion, that of *computational indistinguishability*, defined earlier by Goldwasser and Micali [GM84] (this paper will be discussed at length in Chapter 18). It deems two probability distributions computationally indistinguishable if no efficient procedure can tell them apart.

Definition 7.11 (Computational indistinguishability). Let $\mathcal{C}$ be a family of circuits. Two distributions D and D' on n bits are called $(\mathcal{C}, \epsilon)$-*indistinguishable*, if for every $C \in \mathcal{C}$ we have $|C(D) - C(D')| \leq \epsilon$.

Observe that D is pseudo-random iff D and U_n are indistinguishable. The motivation for this definition was cryptography (for a comprehensive text, see Goldreich [Gol04]), and we briefly describe it. In their seminal paper Goldwasser and Micali [GM84] set the formal mathematical foundations of the field. To illustrate the utility of computational indistinguishability, consider the most basic notion in cryptography: a *secret*. But what is the right definition of secrecy, and how can one achieve it? Goldwasser and Micali propose the fundamental notion of *probabilistic encryption*, which enables hiding even a single bit, as follows. Their miraculous encryption scheme assigns to each of the bits 0 and 1 probability distributions D_0 and D_1 on n bits, with the following two, seemingly contradictory, properties. First, they have disjoint supports, so they are clearly completely distinguishable from an information theoretic perspective. Second, they are

---

[15]But useful, if m is very small.

[16]It is a good exercise to convince yourself that a hardness assumption is needed. Hint: Consider the complexity of determining if a given n-bit sequence is in the image of G, and argue that if G is a pseudo-random generator, then this function must be hard.

computationally indistinguishable, so they look identical to efficient observers. Goldwasser and Micali [GM84] show that not only do such distributions exist, but also how to efficiently generate them. The beauty is that *by definition*, no efficient process can even *guess* the secret bit with probability better than $\frac{1}{2} + \epsilon$. More generally, by repetition, this allows encryption of longer messages in a way that ensures no leakage of *any* partial information. Computational indistinguishability allows the designer of encryption schemes to completely ignore various details about the specific types of adversaries attacking the system and consider only limits on their computational abilities.

Some of you may be wondering at this point how this can be useful for encryption, if *no one* can tell what the secret is. Well, their construction also crucially provides a trap-door that allows "appropriate parties" (with extra knowledge) to be able to tell the two distributions apart efficiently and to decrypt encrypted messages. All this magic relies on a "cryptographic hardness assumption," like one-way functions or trap-door functions mentioned in Section 4.5, which in the paper [GM84], happen to be variants on the hardness of integer factoring. More precisely, and pertaining to a major theme of this book, what [GM84] supplies is a new type of *reduction*. Goldwasser and Micali prove that breaking this encryption scheme (namely, guessing the secret with probability at least $\frac{1}{2} + \epsilon$ from its encryption) will imply a faster algorithm than the hardness assumption allows (e.g., will provide a much faster factoring algorithm than is currently known).

The power of viewing adversaries of cryptographic protocols as computationally bounded entities—and using computational indistinguishability to prove them powerless—is just beginning to be revealed in this example. The paper goes on to develop formal definitions of security of cryptographic protocols that rest on these principles, and they underpin practically every one of the literally thousands of cryptography papers that define and prove properties of cryptographic primitives and protocols. One such example, *zero-knowledge proofs*, is informally explained in Section 10.2 and formally relies on computational indistinguishability. Needless to say, the focus on computational complexity when classifying adversaries fits perfectly with the web of reductions between these numerous primitives and properties, and it allows them to be based on mathematically clean and far better tested hardness assumptions.

Observe the flexibility of the computational indistinguishability definition. It allows any class of adversaries (also called *observers*, *distinguishers*, or *tests*) $\mathcal{C}$, and indeed different settings invite different classes. One particularly interesting class is the one of *all* circuits, or equivalently, all Boolean functions. What does computational indistinguishability of D and D' mean in this case? Simply, it means that for every Boolean function f, we have $|f(C) - f(D')| \leq \epsilon$, which bounds the *statistical distance* of these two distributions (namely, half of the L_1-norm of their difference): $\frac{1}{2}|D - D'|_1 \leq \epsilon$. This illustrates that restricting the class of observers gives a certain *coarsening* of the L_1 metric. It allows two distributions on disjoint supports to be very close in this new metric, and in Section 7.2, we saw that it allows distributions of different entropies to be very close in this metric (even though in both cases, the statistical distance is maximal, essentially 1). This dichotomy between the information theoretic and computational complexity settings is the heart of modern cryptography and will be highlighted again in Chapter 18.

Let us consider one more crucial point. Randomness, the hero of this section, has been used by humans for millennia and has been studied by mathematicians and scientists for centuries. There are various approaches, views, and theories of randomness, in probability theory, statistics, statistical physics, information theory, ergodic theory, chaos theory, Kolmogorov complexity, and indeed, philosophy. This new theory of computational randomness differs from them all in a fundamental aspect. In all past approaches, qualitative or quantitative, the randomness of a phenomenon (be it a coin toss or stock-market fluctuation or a DNA sequence), is an *objective* property of the phenomenon itself. In computational pseudo-randomness, it is *subjective*, a property of the observer! The very same (objective) phenomenon (e.g., the same pseudo-random distribution, or even a single coin toss) can be deemed random by a computationally limited observer, and deemed not random by an observer without computational limitations.

7.3.2 Pseudo-random generators from hard problems

For this section, the readers may want to refresh their memory of Section 4.5 regarding one-way and trap-door functions, as well as circuit complexity from Section 5.2. These two very different sources of hard functions will lead us to two types or paradigms of constructing pseudo-random generators, which differ not

only on the source of hardness they use, but in other important features and some applications as well. The two are sometimes dubbed "BMY-generators" and "NW-generators", and will be discussed in turn.

The phrase "pseudo-random generator" (or PRG for short) was in use decades before these developments. It was used to describe any ad hoc method (particularly those used in practice in a variety of systems and algorithms that need random bits) for deterministically stretching a short sequence into a longer one. Theoretical interest in proving general properties of the output distribution of PRGs probably started with von Neumann [vN51] and his early computer (which needed random bits for Monte-Carlo simulations and weather prediction). A famous quote of von Neumann on the subject is *"Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin."* As is well known, von Neumann ignored his own advice, as shall we.[17]

The idea of constructing pseudo-random generators from hard computational problems developed in a meandering evolution of works. We now recount some of the main ideas, different motivations, and consequences of these works. This high-level description is necessarily sketchy; for more detail, see [Gol99, Gol04, Vad11]. The computational complexity methodology of computational abstraction and efficient reductions, as well as the interconnectivity of areas in computational complexity come out powerfully in this story.

BMY-generators   The first complexity-based definition of a pseudo-random generator was given by Shamir [Sha83]. Again, his motivation was cryptography. He argued that if one user is generating a sequence that another user can *predict*, security and privacy may be violated. He called generators whose output distribution is unpredictable *cryptographically strong* and suggested a design to base this property on a one-way function. Shamir's plan was fully executed by Blum and Micali [BM84]. They formally defined *unpredictable generators*, in which no successive output bit can be nontrivially guessed (with probability $> \frac{1}{2} + \epsilon$), given its predecessors, by *any* efficient observer (in some class $\mathcal{C}$). Let us state these definitions of (left-to-right) unpredictable distributions and generators more precisely. For a distribution D, denote by D_k its projection on the kth bit, and by $D_{[k]}$ its projection on the first k bits.

Definition 7.12 (Pseudo-random generators)**.** Let $\mathcal{C}$ be a family of circuits.

- A distribution D on n bits is called $(\mathcal{C}, \epsilon)$-*unpredictable* (in left-to-right order), if for every $i \in [n]$ and every $C \in \mathcal{C}_{i-1}$, we have $\Pr[C(D_{[i-1]}) = D_i] \leq \frac{1}{2} + \epsilon$.

- A function $G : \{0,1\}^m \to \{0,1\}^n$ is called a $(\mathcal{C}, \epsilon)$-*unpredictable generator* if, on uniformly random input, its output distribution $G(U_m)$ is $(\mathcal{C}, \epsilon)$-unpredictable.

Blum and Micali showed how to construct such efficient unpredictable generators G based on the hardness of the discrete logarithm function (discussed as a one-way function candidate in Section 4.5). The conceptual message here is that some hardness assumptions can guarantee an efficient generation of distributions with a strong pseudo-random property, *unpredictability*, against any efficient observers.

Does this get us closer to our goal of a pseudo-random generator? What is the relation between unpredictable distributions and pseudo-random ones? For example, if a distribution is unpredictable from left-to-right (as above), is it also unpredictable in the other direction? Think about this before reading further.

The answer is "yes"! The next paper in the sequence, by Yao [Yao82b], proved that *every unpredictable distribution is pseudo-random*. More precisely, if D is $(\mathcal{C}, \epsilon)$-unpredictable, then it is also $(\mathcal{C}', \epsilon')$-pseudo-random, with circuits in $\mathcal{C}'$ a little smaller than in $\mathcal{C}$, and with ϵ' a little larger than ϵ. This immediately implies that the Blum-Micali generator is a pseudo-random generator, based on the hardness of the discrete logarithm. In the same paper, Yao extends this result to the wide class of one-way *permutations*.[18] Finally, Yao makes the connection between pseudo-random generators and de-randomization, giving the first (highly) nontrivial de-randomization of $\mathcal{BPP}$ under natural hardness assumptions.

---

[17]Another famous early quote on the subject is the title of Coveyou's paper [Cov69]: "Random number generation is too important to be left to chance."

[18]This is simply a one-way function which happens to be a permutation. Both examples from Section 4.5, modular exponentiation and modular powering, are indeed permutations.

Theorem 7.13 [Yao82b]. *If one-way permutations exist, then for every $\epsilon > 0$, every problem in $\mathcal{BPP}$ can be solved deterministically in deterministic time exponential in n^ϵ.*

Pseudo-random generators based on one-way functions are often called *cryptographic* generators and sometimes "BMY-generators" after their inventors. These generators are extremely efficient (in a sense we'll discuss soon), more than necessary for de-randomization[19] but that is crucial for cryptographic applications. Recall that a one-way function is easy to compute but hard to invert (e.g., modular exponentiation), whose inverse, the discrete logarithm, is assumed hard. This dichotomy is used as follows. The efficiency of the generator depends on the complexity of the easy direction of the function. The class of observers to which its output is pseudo-random depends on the complexity of the hard direction. Thus, this generator can run in polynomial time and can potentially (depending on the assumed strength of the one-way function) withstand super-polynomial or even subexponential time attacks. This is crucial for cryptography, where typical users with their laptops are far weaker than potential adversaries, like companies, governments, or criminals with far larger computational resources. But this ability of cryptographic PRGs to fight stronger adversaries also leads to more conceptual implications, which we now discuss.

First, it is easy to see that a cryptographic PRG implies the existence of one-way functions, almost by definition (inverting it on a sufficiently long part of the output would allow perfect prediction of the rest). Yao proved that one-way *permutations* imply such PRGs, and a sequence of works culminating in a paper by Håstad, Impagliazzo, Levin, and Luby [HILL99], closed this gap. They proved that the two notions are equivalent: One-way functions exist iff cryptographic PRGs do! So, in the cryptographic setting, the most natural hardness assumption and the most natural pseudo-randomness notion coincide.

Theorem 7.14 [HILL99]. *The following are equivalent:*

- *There exist one-way functions.*

- *There exists a $\mathrm{poly}(n)$-time computable $(\mathcal{P}/poly, 1/\mathrm{poly}(n))$-generator $G : \{0,1\}^m \to \{0,1\}^n$ for any $n = \mathrm{poly}(m)$.*

Note that an immediate corollary is the same de-randomization consequence of Theorem 7.13 under the (seemingly) weaker assumption of the existence of one-way functions. Further important consequences of the existence of one-way functions follow from a work by Goldreich, Goldwasser and Micali [GGM86], who constructed pseudo-random *functions*. This is a family of functions that are all easy to compute, but a random one of them cannot be told apart from a truly random function by any efficient observer who can query it at any sequence of (adaptively chosen) inputs.[20] Pseudo-random functions are an extremely strong cryptographic primitive. Besides their obvious utility in a variety of cryptographic settings, I mention two other applications. One consequence of this construction was mentioned in Section 5.2.4—it is essential to the result that *natural proofs* of circuit lower bounds imply quite efficient inversion of all one-way functions (e.g., a much better factoring algorithm than currently known). This may partially explains our inability to prove circuit lower bounds. The second consequence relates to computational learning theory. A central question there is: What classes of functions can be efficiently learned from adaptive observations? (This notion of learning captures, e.g., the acquisition of concepts by children, and the development of the visual system by evolution). Pseudo-random functions, by definition, cannot be learned in this sense, as their value at any new input looks completely random to the (efficient) learner. This observation implies the following, far-from-obvious fact: *Some functions are easy to compute but hard to learn.*

NW-generators   The ability to de-randomize probabilistic algorithms by assuming hardness gave computational complexity a new toy (which indeed became popular): For weak computational models, we *do* have lower bounds, which perhaps can be turned into unconditional pseudo-random generators against them and can unconditionally de-randomize the respective probabilistic classes. One natural candidate was *constant-depth circuits*, where exponential lower bounds were known, by Håstad [Has86] (e.g., for the Parity function).

---

[19]We will soon contrast this with another construction designed specifically for de-randomization.

[20]This may be viewed as a pseudo-random generator with *exponentially* long output (namely, the truth table of the function), every bit of which is efficiently computable, and which the adversary can query in arbitrary locations.

The first to use this hard function directly for pseudo-randomness against this class of circuits were Ajtai and Wigderson in [AW85]. However, their transformation of hardness to pseudo-randomness was quite involved and had rather weak parameters.[21] A few years later, Nisan [Nis91b] proposed a new far more elegant and direct design (explained below) of an unconditional pseudo-random generator based on Parity and its known hardness, again for the same weak class of circuits.

This new design in turn inspired Nisan and Wigderson [NW94] to develop a new *conditional* generator, sometimes called the NW-generator. Their main motivation was weakening the hardness assumptions needed for de-randomizing classes like $\mathcal{BPP}$. As discussed above, the BMY-generators achieve this assuming the existence of one-way functions. If there are no one-way functions, this is a fatal blow to cryptography, but it does not rattle the complexity world much; for example, it leaves $\mathcal{NP}$-complete problems intact (and hard). And indeed, NW-generators *can* utilize even this (less structured) hardness. [NW94] shows how to de-randomize $\mathcal{BPP}$ even assuming the (average-case) hardness, for example, of $\mathcal{NP}$-complete problems. In fact, [NW94] proves a much stronger statement. *Any* function that small circuits cannot approximate, which has an *exponential time* algorithm, yields a pseudo-random generator. Moreover, the construction of the (pseudo-random) generator from the (hard) function is a generic, *black-box* construction.[22] A schematic description of how an NW_f generator is constructed from a given function f is depicted in Figure 19.
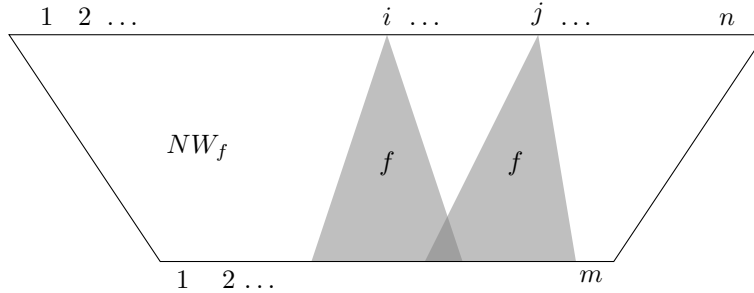
Figure 19. Schematic of NW_f. Essentially, the n outputs are obtained by applying f to n different subsequences (with small pairwise overlaps) of the m-bit long input sequence.

This general result offers a trade-off between the assumed difficulty of the function and the quality of the resulting generator. I state here only one extreme choice of parameters, to contrast with Theorem 7.14. Note that the great relaxation in the hardness assumption is paid for by the running time of the generator (whose stretch and quality do not change). As we shall presently see, this has no effect on de-randomization.

Theorem 7.15 [NW94]. *The following are equivalent statements:*

- *There are exponential-time computable functions that cannot be approximated by polynomial-size circuits.*

- *There exists an $\exp(m)$-time computable $(\mathcal{P}/poly, 1/\text{poly}(n))$-generator $G : \{0,1\}^m \to \{0,1\}^n$ for any $n = \text{poly}(m)$.*

Moreover, instantiating the NW-generator with a function that cannot be approximated by exponential size circuits yields a generator of exponential stretch. As it turns out, this generator provides a nontrivial de-randomization of $\mathcal{BPP}$, despite the exponential running time of the generator. The key observation allowing this is that when using PRGs for de-randomization, one anyway enumerates over all 2^m possible "seeds"

---

[21]Despite this, that type of transformation was resurrected and strengthened 30 years later, together with many new ideas, to obtain much improved parameters and new de-randomization applications. Some of this evolution is in the sequence of papers [GMR+12, CHHL18, RT18], with the last paper demonstrating the power of pseudo-randomness in completely different areas; in this case, quantum computing, the subject of Chapter 11.

[22]This notion means that the construction accesses the function simply by requesting its values on different inputs and is completely independent of the way it might be computed.

of the generator. So we might as well allow, "for free," that the hard function be computable in 2^m-time (rather than, e.g., demand that it is in $\mathcal{P}$), without slowing down the overall deterministic simulation time.

The main weakness of the NW-generator compared to the BMY-generator is that it needs *more* computation time than the adversaries testing the pseudo-randomness of its output. So it is useless for most cryptographic applications. However, for de-randomization purposes, we can afford it, as the generator is only polynomially slower than the adversaries, and for example, for $\mathcal{BPP}$, would still run in polynomial time. There are several advantages of the NW-generator compared to the BMY-generator. The main one of course is that it allows seemingly much more believable hardness assumptions (e.g., of functions complete for $\mathcal{NP}$ and even much higher complexity classes). Another advantage is the generic way in which the hard function is used in the generator—the output bits of the generator are simply evaluations of the hard function on many different, carefully chosen subsets of input bits. This generic construction immediately allows us to de-randomize practically *any* reasonable class of probabilistic algorithms (in other computational models with other resource bounds), assuming a hard function for the class of circuits they correspond to.

The NW-generator also has found far less obvious applications and extensions, for example, in *randomness extraction* [Tre99] (mentioned in Chapter 9), arithmetic complexity [KI04], probabilistically checkable proofs (PCPs) [HW03] and learning theory [CIKK16].

The "optimal" hardness vs. randomness trade-off, Theorem 7.7 of Impagliazzo and Wigderson [IW97], is based on the NW-generator but still needs quite a bit of work beyond it. I mention only one aspect of it. The assumption in this theorem is a standard, worst-case complexity assumption, whereas both the original BMY- and NW-generators used average-case assumptions. The conversion between the two, called *hardness amplification*, transforms any worst-case hard function to one that is hard on the average, and then transforms this one further to another, whose output is essentially unpredictable on a random input. The main tools for hardness amplification are *arithmetization*[23] and the *XOR lemma* (see the history, several proofs, and references in the survey [GNW11]). Getting the optimal parameters for this conversion required a "de-randomization" of such hardness amplification results. Doing so entails building new pseudo-random generators for entirely different purposes than fooling algorithms. This illustrates how concepts like pseudo-randomness evolve and get a life of their own, a phenomenon with many incarnations. Such a pseudo-random generator was designed in [IW97] and led to Theorem 7.7.

7.3.3   Final high-level words on de-randomization

Black-box vs. white-box de-randomization   So far, we have described a "black-box" attack on the de-randomization problem. Namely, one pseudo-random generator is supposed to simultaneously de-randomize *all* probabilistic algorithms from a class, without even looking at them, just by using their input-output behavior. As might be suspected, this very general pseudo-randomness line of attack is a paradigm that benefits from specialization. We saw that to de-randomize a probabilistic algorithm, all we need is a way to efficiently generate a low-entropy distribution that *fools that algorithm only*, on every input, of course. For a *given* algorithm, this may be far easier than fooling simultaneously *all* algorithms. Indeed, such a "white-box" approach, of careful analysis of some important probabilistic algorithms and the way they use their randomness, has enabled making them deterministic via tailor-made generators, *without* any unproven hardness assumptions. These success stories (of which the most dramatic are the recent deterministic primality test of Agrawal, Kayal, and Saxena [AKS04] and the log-space algorithm for undirected graph connectivity of Reingold [Rei08]) actually suggest the route of probabilistic algorithms followed by de-randomization as a paradigm for *deterministic* algorithm design. Many more basic examples of such de-randomization of specific algorithms can be found in Motwani and Raghavan's textbook [MR95]. However, note that for some problems, we cannot expect to eliminate a hardness assumption, even by trying to de-randomize specific algorithms for them! The remarkable result of Kabanets and Impagliazzo [KI04] shows that even de-randomizing the simple probabilistic algorithm embodied in Proposition 7.2 implies certain nontrivial circuit lower bounds. So, hardness and randomness are far more intertwined than anyone expected. Indeed,

---

[23]This term refers to the encoding of Boolean functions as polynomials, an idea developed in the areas of program testing and interactive proofs (some mentioned in Section 10.1)

for the related problem of de-randomizing probabilistic *proofs*,[24] [IKW02] shows that seeing the algorithm doesn't help: white-box de-randomization is equivalent to the black-box one.

Uniform de-randomization   NW-generators are designed to de-randomize probabilistic *non-uniform* circuits. As such, they are expected to require hardness in the form of *non-uniform* circuit lower bounds. But perhaps it is much easier (and requires weaker assumptions) to de-randomize *uniform*, probabilistic algorithms? This has two answers. On the one hand, a certain (on-average) uniform de-randomization is possible under standard *uniform* hardness assumptions (essentially, that $\mathcal{EXP} \neq \mathcal{BPP}$), as demonstrated in [IW98, TV02][25]. On the other hand, for *promise problems*[26], even uniform de-randomization requires circuit lower bounds, as was demonstrated by Tell [Tel18] (see there the interesting development of ideas leading to this result).

Using randomness despite de-randomization   Randomness remains indispensable in many fields of computer science, including probabilistic proofs, learning theory, cryptography and distributed computing, all of which we will meet in subsequent chapters of this book. Moreover, even if $\mathcal{BPP} = \mathcal{P}$, probabilistic algorithms may be simpler and faster. For example, the best known probabilistic primality test takes less than $O(n^3)$ time, while the best known deterministic one requires time $\Omega(n^5)$. In light of all these applications, it is worth pondering: Does perfect randomness as demanded in these applications exist in the real world? Which of these applications would survive if it doesn't? We will return to these questions in Chapter 9.

---

[24]We shall meet many types of these in Chapter 10.

[25]Interestingly, this result still uses the (seemingly non-uniform) NW-generator, but in a more sophisticated way.

[26]Which care only about the answer to some, but not all, problem instances.

8 Abstract pseudo-randomness

In the previous chapter, we saw how the notion of *computational pseudo-randomness* was essential for understanding the power of probabilistic algorithms. Indeed, this notion and its variations, as well as the more general notion of *computational indistinguishability*, have had a huge impact on many aspects of computational complexity, including circuit complexity, computational learning theory, and of course cryptography, where it underlies most definitions and results.

Now let us extend this notion considerably; indeed, take it to its logical conclusion. Rather than consider indistinguishability of distributions from a family of *computationally bounded* observers, we'll simply allow arbitrary families of observers. We will call a property of (any) universe of objects "pseudo-random" with respect to any such family, if these observers cannot distinguish a random object with the property from a random object from the whole universe. Perhaps surprisingly, this will turn out to be extremely useful!

Pseudo-randomness in this general sense is a large and growing area of interaction between the theory of computation and mathematics. In this chapter, I motivate its study from the (different) viewpoints of both fields, give many examples of pseudo-random properties, define this notion in full generality, and raise the question of explicitly exhibiting and certifying pseudo-random objects. We will see that central questions in both fields, including the Riemann hypothesis and the $\mathcal{P}$ vs. $\mathcal{NP}$ question, can be cast very naturally in this language of pseudo-randomness. We will then discuss the emerging proof technique of "structure vs. pseudo-randomness," which already has many applications in diverse areas, but whose power is only beginning to unfold.

Let us start with a few simple examples of pseudo-random properties. These will serve also to highlight a related topic, the *probabilistic method* (for an excellent text, see Alon and Spencer [AS00]). This method proves the *existence* of objects having desired properties without explicitly describing them. This technique was first used in the late 1940s in two independent papers. One is by Erdős [Erd47], proving the existence of Ramsey graphs (and initiating modern *Ramsey theory*), and the other is by Shannon [Sha48], proving the existence of good error-correcting codes (and inaugurating *information theory*). Further examples of Erdős clarified the power of this technique. The essence of this method is (cleverly) picking a universe of objects U and proving that a *random* object in U has the desired property with positive probability (indeed, in most cases, almost surely). Observe that the correctness proof of every probabilistic algorithm has essentially this structure—proving that almost all choices of coin-flips will lead the algorithm to the correct result.

8.1 Motivating examples

We look at three examples of pseudo-random properties. For each, consider the challenge that will occupy us below—to explicitly exhibit objects with these properties.

We start with two examples[1] from Ramsey theory (see Graham, Rothschild, and Spencer [GRS90]). The prototypical meta-theorem of this area is that, in *every* "large enough" system, there must be a nontrivial part that is very structured. The quantitative aspect is how large this subsystem can be.

Ramsey graphs. A graph on n vertices is called "r-Ramsey," if it contains no clique and no independent set of size r. In other words, every subset of r vertices has at least one pair connected by an edge, and another pair that is not connected. Erdős [Erd47] proved that *almost every* graph on n vertices is $(3\log n)$-Ramsey.[2]

Weak tournaments. A *tournament* is a directed graph in which every pair of vertices has a directed edge between them (e.g., describing the winner of pairwise games in a real tournament). A set of vertices (players) is called *dominating* if every other player lost to at least one of them. A tournament on n vertices is called w-weak if it contains no dominating set of size w. Erdős proved that *almost every* tournament is $(\frac{1}{3}\log n)$-weak.[3]

---

[1]While combinatorial in nature, the origins of both examples is in first-order logic, the first leading to a decidability result in Ramsey's original paper [Ram30], and the other to extension theorems and 0/1-laws.

[2]This result is nearly tight. No graph can be $(\frac{1}{2}\log n)$-Ramsey.

[3]This result is nearly tight. No tournament can be $(\log n)$-weak.

For the next example, we turn to coding theory, which underpins our ability to handle noise in digital communication and media storage.

Good codes. A subspace V of dimension (say) $n/10$ over $\mathbb{F}_2^n$ is a *distance-d* linear code if every two vectors in V differ in at least d coordinates. Following Shannon, Varshamov [Var57] proved (via a simple probabilistic argument) that *almost every* such subspace is a distance-$n/10$ linear code.[4]

8.2   General pseudo-random properties and finding hay in haystacks

Let us abstract from the examples above. We have a finite universe of objects U. In the examples above, these objects are all graphs, tournaments, or subspaces, respectively. A *property* is simply a subset $S \subseteq U$ (an element $x \in U$ has the property S if and only if it belongs to S). The properties for our three examples are being r-Ramsey, w-weak, or distance-d, respectively. In all three cases, particular choices of the relevant parameter define properties, which hold for *almost every* object in the respective universe. Note that in all examples, each property was defined by a collection of many "basic tests," which may be viewed as a family of observers. Each of these tests is satisfied with extremely high probability for a random[5] object in U (and typically the proof that all of them are satisfied follows from a union bound). In the first two examples, each such test involves only a small subset of the vertices, and in the third example, each test involves one pair of vectors. This locality, or simplicity, or "low complexity" of basic tests, which measures how "random looking" an object is, will be typical in many other examples considered below.

We will generally say that the property S is *pseudo-random* if it contains *almost all* elements of U. The reason for this name is simply that a random element of U satisfies S almost surely.

Definition 8.1 (Pseudo-random property). A property $S \subset U$ is ϵ-pseudo-random if $|S| \geq (1 - \epsilon)|U|$.

So a pseudo-random property is simply a large set. In particular, it is clear what to do if you need an object with a particular pseudo-random property—simply pick an object in U *at random*, and it will have the required property with high probability. Things get interesting when we want to *deterministically* obtain one. In many cases, it is even difficult to certify that a given object has the property. As usual, we'll think asymptotically; U and S will be families, $U = \{U_n\}$ and $S = \{S_n\}$. Furthermore, as in the examples above, U_n will typically have size exponential in n, so a brute-force search is prohibitive (whereas the description of any particular object takes only poly(n) bits). The value of ϵ, typically unspecified, can be taken to be a small constant, or (as in the examples above) a function $\epsilon(n)$ that tends to 0 as n increases.

This general setting turns out to capture a host of problems in a surprising variety of areas in mathematics and in the theory of computation. In both fields, for a variety of pseudo-random properties $S \subseteq U$, the same questions arise. Does a (mathematically natural) object $x_0 \in U$ satisfy S? Can we efficiently describe some object x in S? Beautifully conveying the essence of such a challenge, Howard Karloff describes it as **finding hay in a haystack!** In many of these problems, despite the abundance of hay, known efficient algorithms may produce only *needles*. However, this quest for hay is well rewarded!

In the following sections, we will establish the generality and importance of this pseudo-randomness phenomenon in math and CS through a series of examples. Before doing so, let us review the three examples of pseudo-random properties above (in reverse order), check that they suit the general framework, and consider the status of these questions for them. In all three examples, the objects in the respective universes U_n can be encoded by about n^2 bits (and so these universes have size $\exp(n^2)$).

Good codes. Shannon's seminal paper [Sha48] left open how to explicitly describe good codes. The search for hay in this haystack is complicated by the fact that in practice, not only do we want a good linear code, but we also need to have efficient encoding and decoding algorithms for it. The study of this collection of questions created the field of coding theory. The first efficient constructions of good linear codes with these

---

[4]This result is nearly tight. No such subspace has distance $n/3$.

[5]We implicitly use the uniform distribution over the set U. Much of the theory and many applications also work with other, and sometimes all, choices of distribution over U. Indeed, in some cases, this applies also to infinite universes endowed with appropriate probability measures.

extra properties were given by Forney [For66] and Justesen [Jus72]. Now we have many alternative ways of constructing a variety of explicit, efficient codes (see, e.g., the textbooks [Rot06, GRS16]). To see that this setting conforms to our notation, note that a subspace of dimension $n/10$ can be described by a basis and so requires $O(n^2)$ bits, the universe U_n is the collection of all of these, and the property $S_n \subseteq U_n$ includes all subspaces with distance $n/10$, namely, all good linear codes.

Weak tournaments. This example also has a happy ending. A fully directed graph on n vertices can be described in $O(n^2)$ bits, U_n the set of all these tournaments, and S_n the tournaments that are w-weak, for $w = \frac{1}{3} \log n$. An explicit pseudo-random object in this setting, which we will call the *Paley tournament*, was suggested by Graham and Spencer [GS71], based on a construction of Paley [Pal33]. Its description is extremely simple. Assume for simplicity that $n = p$ is a prime, with $p \equiv 3 \mod 4$, and let the vertices be the elements of $\mathbb{F}_p$. Recall that $\chi(k)$ denotes the quadratic character function on $\mathbb{F}_p^*$, which is 1 if k is a square in the field and -1 if not. Now for any pair of vertices i, j, direct the edge between them from i to j iff $\chi(i - j) = 1$ (this is a well-defined tournament for such primes as $\chi(k) = -\chi(-k)$). All computations involved here are easy, and so the Paley tournament can be constructed in polynomial time.

While the construction is simple, the analysis uses a deep result: Weil's estimate for the number of rational points in curves over finite fields [Wei49]. The form in which it is used, an *exponential sum* bound, is itself a typical statement of pseudo-randomness, which exhibits the quadratic character χ itself as a pseudo-random object. We will discuss it further in Section 8.3 on the Riemann hypothesis.

Ramsey graphs. Here the journey toward good explicit constructions has taken more than 70 years. Let me summarize what is known. First, as before, n-vertex graphs have $O(n^2)$ bit representations, and U_n is the set of them all. We defined S_n to be the property of being $(3 \log n)$-Ramsey. Exhibiting one such graph remains an elusive problem. It is thus natural to seek pseudo-random objects with weaker parameters, namely, r-Ramsey graphs for larger values of r (of course, one can formally define properties S_n^r as being r-Ramsey, which are all pseudo-random for $r \geq 3 \log n$). Even constructing n^α-Ramsey graphs for small $\alpha > 0$ is nontrivial, and Frankl and Wilson [FW81] give a beautiful construction that is r-Ramsey for $r = \exp(\sqrt{\log n})$. Improvements came from a very different source. The theory of *randomness extractors*, a central type of pseudo-random object we discuss in Chapter 9, has generated a sequence of very different (and less elegant) explicit constructions, first by [BKS+05, BRSW12] (based on arithmetic combinatorics), and then following the breakthrough of [CZ16] (see also [Coh17]) using a very different and intricate combination of methods. These have significantly better parameters; the current best ones yield $r = \exp(\log \log n)^{O(1)}$, quite close (and going down) toward the optimal bound.

8.3 The Riemann hypothesis

The Riemann hypothesis, perhaps the most famous open problem in mathematics, is nontrivial to formally state. The usual formulation involves the *zeta function*, a complex object that requires some advanced, specialized knowledge. Here we present another (known) formulation, in the language of pseudo-randomness, which is elementary and appealing to state and explain even to high-schoolers.

First, let us discuss the *drunkard's walk* (more formally known as the random walk on the integers). Assume there is a pub at 0, and, after having a few beers, a drunkard starts walking randomly up and down the street. More precisely, when occupying an integer i, the drunkard moves to $i + 1$ with probability $\frac{1}{3}$, to $i - 1$ with probability $\frac{1}{3}$, and stays at i with probability $\frac{1}{3}$. How far from the pub is he likely to be after n steps? This can be formulated as estimating the sum of a sequence of n independent, unbiased random $\{-1, 0, 1\}$ variables. It is a standard calculation to prove that he will be within $O(\sqrt{n})$ distance of the pub with high probability.

This suggests a pseudo-random property. The universe is $U_n = \{-1, 0, 1\}^n$, consisting of all possible n-walks. Define a walk $z \in U_n$ to be *d-homebound* if it ends up within d of the pub, namely, $|\sum_i z_i| \leq d$. As mentioned, being $d = d(n)$-homebound is a pseudo-random property for any $d(n) \geq c\sqrt{n}$. Can we find an explicit sequence with this property? Sure we can; there are many easy ones, like the all-0 sequence, or any sequence with an equal number of 1's and -1's. The interesting question to ask here is whether any natural

mathematical sequences possess that "square-root" cancellation, so typical of random sequences. Here is one famous natural sequence.

Definition 8.2 (Möbius sequence). Define (the infinite) *Möbius sequence* $\mu = \mu(k)$ for every natural number k as follows: $\mu(k)$ is 0 if k has a square divisor. Otherwise, it is -1 or 1, depending on whether k has an odd or even number of prime divisors, respectively. Define μ_n to be the first n symbols of μ.

If our drunkard marched according to the instructions of the Möbius sequence, would the sequence always stay d-homebound for d around $\sqrt{n}$? This simple question about this simply defined sequence is equivalent to the Riemann hypothesis. More precisely, Mertens [Mer97] proved the following theorem.

Theorem 8.3 [Mer97]. *The Riemann hypothesis is true if and only if, for every $\delta > 0$, the sequence μ_n is $n^{\frac{1}{2}+\delta}$-homebound.*

Of course, the Riemann hypothesis is not known to hold. So it is natural to weaken the pseudo-randomness demand (as we did for Ramsey graphs), and ask, for example, for *any* nontrivial cancellations. Namely, is μ_n at least $o(n)$-homebound? This innocent question also turns out to be equivalent to a well-known statement, which in this case is a theorem rather than a conjecture. Namely, it is equivalent to the celebrated Prime Number theorem of Hadamard and de la Vallé-Poussin, which determines the asymptotics of the number of primes below an integer x to be $x / \ln x$.

Theorem 8.4 (Prime Number Theorem). *The sequence μ_n is $o(n)$-homebound.*

How close is the Möbius sequence to a sequence of coin tosses? Well, it is of course deterministic and hence completely predictable. Its symbols can be successively computed by a Turing machine. Equivalently, the Möbius sequence correlates perfectly with a sequence produced by some deterministic Turing machine. It now makes sense, in the spirit of this chapter, to subject it to a smaller class of tests than all computable sequences. Let us start from the bottom. The prime number theorem (Theorem 8.4) can be interpreted as saying that the Möbius sequence has vanishing correlation with the absolutely simplest deterministic sequence, the constant sequence $1, 1, 1, \ldots$. How about the alternating sequence $1, -1, 1, -1, 1 \ldots$? How about a sequence produced by a finite automaton, or a real-time Turing machine?[6] A bold conjecture of Sarnak is that the Möbius function has vanishing correlation with *every* sequence that is generated by a dynamical system of zero entropy rate.[7] Some very general cases of this conjecture have been proved (see Bourgain, Sarnak, and Ziegler [BSZ13] for precise definitions and historical survey).

While square root cancellation for the Möbius sequence remains a major question of mathematics, such cancellation was proved for other important sequences in other major theorems. Let's give an example of this with the theorem we needed for the "weak tournaments" example. A consequence of Weil's theorem [Wei49] is the following *exponential sum* bound for the quadratic character χ.

Theorem 8.5 [Wei49]. *For every prime p, every degree $d > 0$, and every nonsquare polynomial $f \in \mathbb{F}_p[x]$ of degree d,*

$$\left| \sum_{x \in \mathbb{F}_p} \chi(f(x)) \right| \leq d\sqrt{p}.$$

Thus, to all these low-degree polynomial "tests," the quadratic character looks as random as a sequence of coin tosses, at least from the viewpoint of home-boundedness. Similar results hold for other characters. More importantly, a *multivariate* polynomial generalization of this theorem follows from Deligne's celebrated Riemann hypothesis for varieties over finite fields [Del74, Del80].

Exponential sums and related estimates pervade number theory, analysis, and ergodic theory, and may also be viewed from this pseudo-randomness angle. While it is not clear if this angle is powerful enough to prove new such results, this connection was extremely fruitful for a variety of applications. As an illustration, Theorem 8.5 was used in [AGHP92] for de-randomization; in [BGW99] for lower bounds; and as we saw above,

---

[6]Such a machine must output a symbol at every computational step.

[7]This is a considerably stronger model than a deterministic real-time Turing machine—it may be viewed as a probabilistic real-time Turing machine with access to $o(n)$ random coins before it outputs the nth symbol.

in [GS71] for weak tournament constructions. Some pseudo-random objects (like the quadratic character) have surprisingly wide applicability, and we shall see a much stronger demonstration of this for expanders and extractors below.

8.4 $\mathcal{P}$ vs. $\mathcal{NP}$

How can the $\mathcal{P}$ vs. $\mathcal{NP}$ question be about pseudo-randomness? The short answer is the following: Almost all functions are hard to compute; is SAT hard to compute? This fits our general framework in perfect analogy with the previous one of the Riemann hypothesis: Almost all sequences are home-bound; is the Möbius sequence home-bound? Elaborating on this analogy, replace the universe of sequences by that of Boolean functions, replace the pseudo-random property of home-boundedness by the pseudo-random property of computational difficulty, replace the Möbius sequence by the SAT function, and you have replaced the Riemann hypothesis with the $\mathcal{P}$ vs. $\mathcal{NP}$ problem as a pseudo-randomness question. We hope that by now the reader can apply this analogy to other settings: All you need are the three ingredients, the universe U, a pseudo-random (= large) property $S \subset U$, and an element $x \in U$ whose membership in S (namely its pseudo-randomness with respect to that property S) is in question. As already mentioned, this setting, clearly or in disguise, captures many diverse mathematical and computer science problems.

Let us return to $\mathcal{P}$ vs. $\mathcal{NP}$ and be a bit more formal, as making this example suit our pseudo-randomness framework requires a particular way of setting the parameters. As we shall see, this viewpoint on $\mathcal{P}$ vs. $\mathcal{NP}$ will also explain the importance of pseudo-randomness to the "natural proof" barrier to lower bounds that we met in Section 5.2.4.

To be consistent with our notation in this section, it will be useful here to call the input size of a Boolean function by k, so we consider functions $f : \{0,1\}^k \to \{0,1\}$. It will also be convenient to define another complexity class, $\mathcal{EXP}$, of all functions $f = \{f_k\}$ computable in $\exp(k)$ time.

Our universe U_n will be all Boolean functions whose *truth table* takes n bits; these are Boolean functions $f : \{0,1\}^k \to \{0,1\}$ with $n = 2^k$. Observe that the truth table of SAT, or for that matter any function in $\mathcal{EXP}$, can be produced in $\exp(k) = \mathrm{poly}(n)$ time. Now, according to Theorem 5.6, almost all functions in U_n require circuit size of at least $2^{k/3} = n^{1/3}$. So the property of a k-bit function f, requiring circuit-size at least h for any $h(n) \leq n^{1/3}$, is pseudo-random. Call this property "being h-hard." By definition, if SAT (or any problem in $\mathcal{NP}$) is h-hard for even any $h(n) \gg (\log n)^{O(1)} = \mathrm{poly}(k)$, it would immediately imply that $\mathcal{NP} \nsubseteq \mathcal{P}/\mathrm{poly}$, resolving the most important open problem in this book! Short of proving it for SAT or another explicit function, an easier task is to efficiently generate such pseudo-random h-hard functions. This, too, is an important challenge of complexity theory, for the following reason. Observe that if we have a $\mathrm{poly}(n)$-time algorithm for generating (given n) the truth table of such an h-hard function f, it means that $f \in \mathcal{EXP}$. Thus, such an algorithm, for $h(n) \gg (\log n)^{O(1)} = \mathrm{poly}(k)$, would imply the following, much weaker, however very important conjecture.

Conjecture 8.6. $\mathcal{EXP} \nsubseteq \mathcal{P}/\mathrm{poly}$.

Summarizing, pseudo-randomness can naturally capture proving conjectured circuit size lower bounds. Curiously, the same pseudo-randomness notion can also explain our difficulty in proving circuit lower bounds (e.g., resolving Conjecture 8.6). As mentioned in Section 5.2.4, if factoring k-bit integers requires circuit size $\exp(k^\epsilon)$, then Conjecture 8.6 has no *natural proof*. We are in a position to better explain this surprising result of Razborov and Rudich [RR97] from Section 5.2.4, now that we have developed the appropriate language of pseudo-randomness. Let us explore what pseudo-randomness has to do with the difficulty of proving computational hardness.

Natural proofs (of circuit lower bounds) certify the computational hardness of Boolean functions, and by definition, do so for *almost all* of them. In other words, having a natural proof is a pseudo-random property. But the assumed hardness of factoring enables, by an important result (discussed after the statement of Theorem 7.14) of Goldreich, Goldwasser and Micali [GGM86], the construction of *efficiently computable* pseudo-random functions. Such functions, by definition "look like" random functions, and in particular satisfy this pseudo-random property. However, by construction, these functions are in fact easy to compute! This contradiction is at the heart of the challenge of proving circuit lower bounds, even beyond the specific

notion of natural proofs. The lower bound, implicitly or explicitly, has to figure out a measure (or criterion) that will distinguish hard random functions from easy pseudo-random ones, even though in a very strong sense, the two collections seem indistinguishable. The Razborov-Rudich notion of natural proofs is one family of measures (or criteria) that is useless for such lower-bound provers (at least assuming that integer factoring is hard, or more generally if any one-way function exists).

8.5   Computational pseudo-randomness and de-randomization

In this section, we will see that the abstract pseudo-randomness framework of this chapter is general enough to capture the concrete *computational* pseudo-randomness discussed in Section 7.2 (which you may want to review). So we can add to the impressive list of major open problems captured by this framework the $\mathcal{BPP} = \mathcal{P}$? problem. After all, the quest to "find hay in a haystack" *efficiently* is precisely the problem of de-randomization. Still, it will be useful to spell out more precisely some of the challenges and results of computational pseudo-randomness in this general framework.

First, consider the de-randomization of specific probabilistic algorithms. Let $A(x, y)$ be a deterministic algorithm that for every input x, on random input y causes A to output the correct answer (for some fixed function on inputs x) with high probability. We would like to find a deterministic algorithm for the same problem. Putting this task into our general framework is simple. We have a universe U_x for every input x, of all possible random sequences y. For each we have a pseudo-random property $S_x \subseteq U_x$ of all sequences y, leading the algorithm to give the correct output on input x. Because the algorithm A succeeds with high probability, S_x contains most elements of U_x. Efficiently finding, for every given x, such pseudo-random $y \in S_x$ will de-randomize the algorithm A. It is important to note that different algorithms for the same problem lead to different "haystacks," and different notions of pseudo-random "hay" to seek; some of these de-randomization tasks may be easier than others.

A great success story of this approach is the deterministic primality test algorithm of Agrawal, Kayal, and Saxena [AKS04]. Indeed, this paper solved such a search problem by de-randomizing one specific probabilistic primality algorithm, that of Agrawal and Biswas [AB03]. Note that this probabilistic algorithm was designed with the hope that the search problem above may have a deterministic algorithm (and it did!). The nature of the haystack arising from this algorithm is too complex to describe here (we give more details in Section 13.1). But even earlier, primality tests provided another haystack, which is simpler to describe (and also to de-randomize) *under a number-theoretic assumption*, as we now discuss. Indeed, the story of efficient primality tests begins with a *deterministic* algorithm. In 1976, Miller [Mil76] gave an efficient deterministic primality test *assuming* the extended Riemann hypothesis. In our general framework, and reversing history, Miller's algorithm may be viewed as a de-randomization of the (later) probabilistic primality test of Rabin [Rab80], who actually designed the test to eliminate the extended Riemann hypothesis assumption underlying Miller's algorithm.[8] The search problem for Rabin's algorithm is much easier to describe (and we will cheat, and discuss one even simpler). Let x be an integer. The haystack U_x may be viewed as all numbers modulo x. A number y in this set is good (i.e., in S_x) if it has Jacobi symbol -1. At least half of the y's are good (i.e., will lead to a correct output for Miller's algorithm). What Miller observed, using a number-theoretic result of Ankeny [Ank52], is that there must be a good y among the first n^2 integers, where n is the binary length of x. Thus, a simple search through all small integers solves the search problem (finds hay) in deterministic polynomial time (assuming the extended Riemann hypothesis, of course). In other words, the set Y of the first n^2 integers is (when considered modulo x) a perfect sample: It will contain a good y for every U_x with n-bit input x.

Next, we reformulate our general framework to capture the de-randomization result for $\mathcal{BPP}$ of Section 7.2 (which does require hardness assumptions). The goal is to simultaneously fool *every* efficient probabilistic algorithm A on *every* possible input x. To do so, let us design an efficient pseudo-random generator, which for our purposes here is captured by its image: a polynomially small collection Y of n-bit sequences (say, $Y = \{y_1, y_2, \ldots, y_t\}$ with $t \leq \text{poly}(n)$), which is a "nearly perfect sample set." Namely, the uniform

---

[8]Solovay and Strassen [SS77] designed a somewhat different probabilistic primality test earlier, independently of Miller and Rabin.

distribution on this sample Y is computationally pseudo-random: It looks like the uniform distribution on *all* sequences, to every small circuit.[9] The de-randomization procedure for any algorithm A and input x will simply take a majority vote of $A(x, y_i)$ over $y_i \in Y$. The pseudo-random property of Y guarantees that most of its elements cause A to output the correct answer, and so the majority vote will always be correct.

Let us spell out the ingredients of this approach to the $\mathcal{BPP} = \mathcal{P}$? question more precisely with concrete parameters in our general framework. Let $t = n^4$.

- The universe U_n is the set of all t-subsets of $\{0, 1\}^n$.

- The (pseudo-random) subset S_n contains all $Y \in U_n$ such that the uniform distribution on Y is pseudo-random.[10]

- The set S_n contains almost every Y in U_n, and so is indeed pseudo-random. This can be established by a standard counting argument.

- The related search problem is: Given n, find an element of S_n.

- If the search problem has a poly(n) time algorithm, then $\mathcal{BPP} = \mathcal{P}$.

The gist of the general de-randomization results of Section 7.2 is the design of such an efficient algorithm for this search problem, assuming a hard enough function exists. The pseudo-random set Y constructed by this algorithm is the image of an efficient pseudo-random generator based on such a hard function.

Is it possible to get such general de-randomization results unconditionally? For polynomial time circuits, efficient pseudo-random generators imply circuit lower bounds that we cannot currently prove (in the terminology of this section, the set of *tests* is simply too powerful). So here, hardness assumptions are necessary for pseudo-randomness. But following this paradigm has led complexity theorists to consider a large variety of interesting computational models that are weaker than polynomial-size circuits. These models compute fewer functions, so this battery of tests is smaller, which makes the pseudo-random property larger and the potential to find a pseudo-random object easier. This idea has been extremely fruitful and led to *unconditional* pseudo-random generators for a wide variety of important classes, including memory-bounded algorithms, constant-depth circuits, and low-degree polynomials.

We conclude with one celebrated example of such work, which will be discussed again in Chapter 14. Consider probabilistic algorithms that use little memory—that are *logarithmic* in the input length. A relevant example is an algorithm that performs a random walk on a graph; it needs only to remember at every stage the name of the vertex it currently occupies, whose binary length is logarithmic in the size of the entire graph. Now let us consider to what kind of tests such algorithms can subject their random input. Counting is one type of computation that is easily performed by limited-memory algorithms. Therefore, they can perform many standard statistical tests appearing in statistics textbooks and used in numerous scientific experiments. Typically such tests count various small patterns in the sequence and check that they are distributed roughly as in a random sequence. The design of (nearly perfect) sample sets, even for specific statistical tests, occupies the field of experimental design in statistics. If we aim to fool all limited-memory algorithms, we in particular ask: Can one fool *all* these tests simultaneously and unconditionally? Remarkably, a definite positive answer was provided in the seminal work by Nisan [Nis92], who devised a beautiful low-memory pseudo-random generator against all such algorithms. This generator uses slightly super-polynomial time (a more precise statement appears in Chapter 14 on space complexity).

While explaining these important results is beyond the scope of this text, note that also here there is a source of hardness which implies pseudo-randomness, once it is encapsulated properly. Nisan's insight[11] is that a good way to encapsulate pseudo-randomness tests performed by small memory algorithms is by low communication 2-party protocols (so the primary resource, communication, is actually information theoretic rather than computational). More precisely, one needs to establish that computing (or even approximating)

---

[9]These circuits capture the computation of all efficient algorithms A on all inputs x of a given length.

[10]In the sense of Chapter 7.2, with respect to all n^3-size circuits. Namely, every such set Y is a nearly-perfect sample set for every algorithm A running in time n^2 using n random bits.

[11]Which was sparked by a homework problem in a complexity class of Umesh Vazirani that he took at Berkeley.

certain functions on two arguments $h(x, y)$ requires a lot of communication between two parties, one of which is holding x and the other y. While this is easy to prove, converting it into a pseudo-random generator requires more ideas and work! This viewpoint of Nisan's generator is explained and expanded in [INW94]. The field studying these kinds of lower bounds is called *communication complexity*, discussed in Chapter 15.

8.6   Quasi-random graphs

In this section and the next one, we study graphs. This section focuses on "dense" graphs (with quadratically many edges) and the next on "sparse" graphs (with linearly many edges). The theory of *quasi-random graphs* originated in the papers of Thomason [Tho87] (who actually called them "pseudo-random graphs") and Chung, Graham, and Wilson [CGW89] (whose terminology of "quasi-random graphs" stuck). It is one of the earliest examples of a comprehensive study of pseudo-random properties in the sense we discussed here and illuminates a few points we have not yet addressed, specifically, reductions and completeness for such properties.

The study of random graphs and their properties was initiated in the seminal papers of Erdős and Rényi [ER59, ER60] and became a huge field of inquiry. Recall that a random graph on n vertices is defined by letting every pair of vertices have an edge between them with probability $\frac{1}{2}$. In other words, it is the uniform distribution on the set, which we naturally name U_n, of all undirected graphs on n vertices.[12] Consider several different properties, all of which are quite easy to prove are pseudo-random (namely, to hold for almost all graphs $G \in U_n$). Note that the first three address seemingly different aspects of a graph G; the first counts the number of edges in large subsets of G, the second counts the occurrences of small "pattern" graphs in G, and the third computes an algebraic property—the top eigenvalues of G's adjacency matrix. In all, these parameters are required to be close to their expectation values in a random graph, asymptotically as n grows.

S_1: For *every* subset of vertices $T \subset [n]$, the number of edges in T is $|T|^2/4 \pm o(n^2)$.

S_2: For *every* fixed *labeled* graph H, the number of induced copies of H in G is $(1 \pm o(1))n^v 2^{-\binom{v}{2}}$, where v is the number of vertices in H.

S_3: The largest two eigenvalues of the adjacency matrix of the graph G satisfy $\lambda_1 = (1 \pm o(1))n/2$ and $\lambda_2 = o(n)$.

S_4: The number of edges in G is $(1 \pm o(1))n^2/4$, and the number of 4-cycles in G is $(1 \pm o(1))n^4/16$.

Chung, Graham, and Wilson's paper [CGW89] proves the following remarkable statement: all four properties are equivalent.

Theorem 8.7 [CGW89]. *If a (large enough) graph satisfies any one of these properties, it satisfies them all.*

Indeed, any graph satisfying one of these properties, satisfies many others studied in that paper. This suggests some notion of *completeness* for pseudo-randomness, which is indeed brought out most powerfully by the last property, S_4. Note that S_4 tests only two parameters of the graph, whereas S_2 tests these as well as an unbounded number of others. While it is obvious that a graph satisfying S_2 also satisfies S_4, the surprising fact is that the converse holds as well. So, the statistics of edge and 4-cycle occurrences in a given large graph dictates (up to negligible error terms) the statistics of every finite subgraph!

The paper [CGW89] also studies which specific graphs are pseudo-random in the sense above. The reader might not be stunned to find out that one answer is the canonical example, the Paley graph. This is actually a variant of the Paley tournament we saw in Section 8.2 above, in which the number of vertices is a prime $n = p$ but this time with $p \equiv 1 \mod 4$, with an edge between i and j iff $\chi(i - j) = 1$ (which is well defined, since for such p, we have $\chi(k) = \chi(-k)$). To prove its pseudo-randomness it suffices to do so for the simplest

---

[12]To have a fully consistent notation, we could have named this set $U_{\binom{n}{2}}$, as the bits describing this distribution are the edges of the graph. But I expect no confusion should arise.

property, which is S_4. And this property holds (using Weil's Theorem 8.5 again), since every pair of vertices has $n/4 \pm O(\sqrt{n})$ common neighbors, from which the counts follow.

Of course, one can study other classes of random graphs and their properties under different distributions. Interestingly, one can also go in the reverse direction: Start from the (would be) pseudo-random properties and develop classes of random graphs (or other objects) from them. A case in point, which developed into an important theory is the following. Take any sequence of graphs $G = \{G_n\}$ for which all statistics in S_2 converge in an appropriate natural sense (namely, the graphs G_n share, in the limit, the occurrence frequencies of all finite graphs H). Then this sequence gives rise to a random graph model (called a *graphon*) that greatly generalizes the Erdős-Rényi theory, and from which one can sample graphs of any size in a natural way (which we will not consider here). In this model, the property of having these specific subgraph statistics is a pseudo-random property! This emerging theory of *graph limits* is extremely exciting. It connects to algorithms and coding theory through the area of *property testing* (see Goldreich [Gol10]), and to statistical physics through the study of partition functions and Gibbs distributions. Finally, it allows doing analysis and using analytic tools (in the classical sense of limits, convergence, compactness, etc.) in the combinatorial area of extremal graph theory. The reader is encouraged to find out more (see Lovász's book [Lov12]).

8.7 Expanders

Expander graphs are perhaps the most universally applicable pseudo-random objects. They play key roles in almost every area of the theory of computation: algorithms, data structures, circuit complexity, derandomization, error-correcting codes, network design, and more (indeed, they appear in many chapters of this book). In mathematics, they touch in fundamental ways different subareas in analysis, geometry, topology, algebra, number theory, and of course graph theory. Indeed, expanders are an important bridge between complexity theory and mathematics; Chapter 13, devoted to these interactions, describes four distinct areas where graph expansion plays a key role (see Sections 13.3, 13.4, 13.5, 13.6).

Precious few nontrivial mathematical objects can boast a similar impact! Hoory, Linial, and Wigderson's monograph [HLW06] is a comprehensive text on expanders. However, this field is extremely active and much has happened since it was published; some of the exciting recent developments are summarized here.

Expanders are sparse graphs. So here we will consider our universe U_n to be all d-regular graphs on n vertices (namely, every vertex is touched by d edges), where d is the same fixed constant for all n. Here are some natural pseudo-random properties of these objects. I state them informally. Note again that they seem entirely different, one expressing a combinatorial/geometric property, the second an algebraic property, and the third a probabilistic property, of a d-regular graph.

S_1: For every t and subset of vertices $T \subseteq [n]$ of size $|T| = t$, the number of edges between T and its complement is roughly $dt(n-t)/n$.

S_2: All nontrivial eigenvalues of the adjacency matrix of G are bounded away (in absolute value) from the first one, which is d.

S_3: The natural random walk on G converges to the uniform distribution in $O(\log n)$ steps (at an exponential rate).

It is standard to check that each of these properties holds for almost every d-regular graph. Again, here we have the surprising result that all three properties are equivalent: If any graph has one, it has the others (a sure sign that the notion defined is *basic*)! The equivalences are known with specific quantitative relations between the different unspecified parameters, through a series of works in the 1980s [Tan84, AM85, Alo86, SJ89] (the connection between S_1 and S_2 is a discrete analog of the important *Cheeger inequality* for Riemannian manifolds [Che70]).

A graph is an expander if it is pseudo-random in this sense. The first to define expanders and prove their existence (via the probabilistic method) was Pinsker [Pin73]. Can one explicitly construct expanders? In Section 8.6, a pseudo-random graph presented itself: the Paley graph. In the present sparse setting, the

construction is far less obvious. The problem of explicitly constructing expanders has attracted researchers and techniques from many different fields. The first explicit construction is due to Margulis [Mar73], who used the "Kazhdan property (T)." Today we know of a variety of ways, algebraic and combinatorial, to construct expanders (the main approaches are listed at the end of the section). Precise definitions, different constructions, and many applications appear in the monograph [HLW06]. More applications are sketched in the survey talk available in Wigderson [Wig10a].

Let us give some detail on one construction, one open problem, and one application. These should tempt the reader to find out more about these remarkable objects.

An explicit family of expanders   Let p be a prime, and consider the 3-regular graph G_p whose vertices are the elements of $\mathbb{F}_p$, and connect every vertex to its predecessor, successor, and inverse. In other words, every x is connected by an edge to $x-1, x+1$, and x^{-1} (as 0 has no inverse, we can connect it to itself). The following theorem is from Section 3.3 in Sarnak [Sar90].

Theorem 8.8 [Sar90]. *The family G_p is a family of expanders.*

While the graphs themselves are simple to describe, their expansion proof uses very sophisticated tools (as is common in this field). It follows from the expansion of the Cayley graphs[13] on $SL(2, p)$ with the standard generators, and the fact that the graphs G_p are Schreier graphs of the action of these groups on the projective line by the Möbius transformation. The expansion of $SL(2, p)$ was first derived from Selberg's famous 3/16-theorem in number theory [Sel65]. A different proof of this expansion, using arithmetic combinatorics, was given by Bourgain and Gamburd [BG08].

Let us observe how explicit these graphs G_p are. If p is an n-bit integer, we can represent the elements of $\mathbb{F}_p$ by n-bit sequences. The neighborhood structure is so simple that, given a vertex x, it is possible to compute its 3 neighbors in $\text{poly}(n)$-time. So we have an exponential size graph that has such a succinct description, by the algorithm for neighbors. This level of explicitness[14] will be crucial in the application we present, and it turns out to be crucial in many others. We give one convenient description of how explicit and efficient these constructions can be. The parameters of the expanders chosen here are not the most general, but they are convenient and suffice for most applications.[15]

Theorem 8.9. *For every constant c, there is a constant d and a $\text{poly}(n)$ time[16] algorithm A, such that for every integer n, there is a d-regular graph G_N on $N = 2^n$ vertices[17] with the following properties.*

- **Explicitness:** *On inputs n and $x \in \{0,1\}^n$, A outputs the d neighbors of x in G_N.*

- **Eigenvalue expansion:** *All nontrivial eigenvalues of G_N are bounded above in absolute value by $d/2$.*

- **Vertex expansion:** *Every set $S \subseteq \{0,1\}^n$ of size $s \leq o(N/d)$ has at least cs neighbors in G_N.*

A note on the optimality of parameters will lead us to our open question. For the eigenvalue definition of expansion, the optimal relationship between c and d is known and can be achieved. By the Alon-Boppana theorem [Alo86] (see proof in [Nil91]), $c \geq 2\sqrt{d-1} - o(1)$. This was achieved by the explicit Ramanujan graphs of Lubotzky, Phillips, and Sarnak [LPS88] and of Margulis [Mar88], which satisfy $c \geq 2\sqrt{d-1}$. However, for *vertex expansion*, random graphs satisfy "lossless expansion"[18] $c \geq (1 - o(1))d$, whereas the best known explicit construction of Kahale [Kah95] achieves "only" $c \geq (\frac{1}{2} - o(1))d$. Explicitly achieving $c > d/2$ has been open now for 20 years. The best partial result is a construction of *bipartite* graphs that

---

[13]The vertices of Cayley graphs are all elements of a given group, and two vertices are connected if their ratio belongs to a given set of generators.

[14]Actually, efficiently obtaining large primes p may be difficult, as we mentioned in Section 7.1, and so the description here is not fully explicit. However there are ways around this problem (which we don't discuss) that result in fully explicit graphs with the same properties.

[15]Some applications demand even more stringent explicitness and efficiency, and it is actually remarkable how cheaply these complex pseudo-random graphs can be generated — see e.g., [VW18].

[16]Indeed, even logarithmic space.

[17]We pick a power of two so as to label vertices by binary sequences, but one can pick any other integer.

[18]They have essentially as many neighbors as possible, as this number cannot exceed sd.

expand losslessly in one direction [CRVW02]), a property that already suffices for applications beyond what eigenvalue expansion achieves.

Open Problem 8.10. Construct explicit lossless expanders (say with $c > .9d$).

An application of expanders to randomness-efficient error reduction   For our application, consider the following problem, which may be called *deterministic error reduction*. You have a probabilistic algorithm A that you want to run on input x. This requires n random bits, which is exactly what you possess. The problem is that the error guaranteed by the algorithm is, say, $1/10$, which is far too high for you. So you would like to reduce it. We showed in the probabilistic algorithms in Section 7.1 that errors can be easily reduced (e.g., to $\exp(-k)$ for any k). The idea is to run A on x for k times with independent randomness each time and then take the majority vote of the answers. However, this requires kn random bits, which you don't have. Is any reduction of error possible with only the n random bits you have? A beautiful positive answer was given by Karp, Pippenger, and Sipser [KPS85], which is one of the earliest applications of expanders. Simply use your random bits to produce a random vertex x in the graph G_p above.[19] Then consider all vertices at distance $d = O(\log n)$ from x, where each of them is an n-bit sequence. Use each of them as randomness when running A on x, and as before, compute the majority vote of the answers. Note that the process of finding all these vertices is efficient—simply apply the algorithm computing neighbors in G_p repeatedly to generate all $3^d = \text{poly}(n)$ paths in G_p and take their endpoints. What is less obvious, but follows from the expansion properties of G_p, is that the error of this algorithm will reduce to any n^{-c} (the choice of c determines the constant in the definition of d).

Methods of proving expansion   Expansion is not only a fundamental and widely applicable notion across mathematics and computer science, but also has remarkably diverse sources. By now we have a surprising wealth of methods to explicitly (and nonexplicitly) construct expander graphs, each with its own benefits and consequences. In particular, these methods give a comprehensive, if still incomplete, understanding of the broad challenge set by Lubotzky and Weiss [LW92]: Find out which finite groups, with which generating sets, yield expanding Cayley graphs. I say a few words about each, which may inspire the reader to dig deeper.

- The "mother group" approach, initiated by Margulis [Mar73]. Here the family of finite expanders is made up of Cayley graphs, and the underlying groups are all quotients of a single infinite group. Properties of this mother group determine the expansion of the quotients. This approach led to the eigenvalue-optimal *Ramanujan expanders* of [LPS88, Mar88], mentioned above.

- The "bounded generation" approach, initiated by Shalom [Sha99]. It leads (with many other ideas) to Kassabov, Lubotzky, and Nikolov's [KLN06] very general theorem that every[20] (non-Abelian) finite simple group has a fixed set of generators that make the Cayley graph expanding.

- The "zig-zag" product approach, initiated by Reingold, Vadhan, and Wigderson [RVW02]. This combinatorial method iteratively constructs larger and larger expanders from a fixed one. A connection of this combinatorial method to semi-direct product in groups [ALW01] has led to Cayley expanders of some very nonsimple groups [MW04, RSW04]. The zig-zag method underlies the construction of lossless bipartite expanders [CRVW02] mentioned above. It has also led to a breakthrough in computational complexity: Reingold's proof [Rei08] that $\mathcal{SL} = \mathcal{L}$ (see Section 14), which in turn inspired Dinur's combinatorial proof [Din07] of the PCP theorem (see Section 10.3).

- The "arithmetic combinatorics" approach, initiated by Bourgain and Gamburd [BG08]. Here again, expanders are Cayley graphs. Expansion follows (among other important ideas) from growth of sets under group product, starting with Helfgott [Hel08], which in turn is based on the "Sum-Product theorem"

---

[19]Assume here for simplicity that $p = 2^n$ (which is, of course, impossible). The case $p \neq 2^n$ can be handled as well with some care.

[20]The missing Suzuki group was added to complete this list in [BGT10].

in finite fields of Bourgain, Katz and Tao [BKT04]). This general method works to prove expansion in *all* simple linear groups of finite rank, with almost every pair of generators (as opposed to specially chosen ones in other methods) [BGGT13]. This powerful method also underlies expansion in unitary groups [BG10], the explicit construction of *monotone expanders* and *dimension expanders* [BY13], and the *affine sieve* [BGS10], among other applications.

- The "lifting" approach, initiated by Bilu and Linial [BL06]. Again, this is a combinatorial, iterative method, where larger expanders are generated from smaller ones via lifting. Optimal analysis via "interlacing polynomials" by Marcus, Spielman, and Srivastava [MSS13a] of this constructive lifting method has led to completely new Ramanujan expanders (made constructive in [Coh16]). It also had other major consequences, most notably the resolution of the Kadison-Singer conjecture, discussed in Section 13.3.

Variations and extensions   The definition of expansion discussed above is perhaps the most studied and used, but is just one of many useful and interesting ones. We discuss briefly two others, which are hot research topics producing beautiful structural, algorithmic and applicative theory.

The first is *small set expansion*. The definition above focuses on the expansion of large sets. But the expansion *profile* of a graph (see [HLW06]), namely the edge and vertex boundaries of subsets of every given size, is natural to study, and one expects smaller sets to have larger expansion. The algorithmic and complexity theoretic importance of small set expansion, especially with respect to understanding hardness of approximation, started revealing itself in works on UGC (the unique games conjecture) of Khot [Kho02], discussed in Section 4.3.2. In particular, the resolution of the somewhat weaker conjecture on "2-2 games" crucially depends on understanding the expansion of small sets of the *Grassmann graph* (see [KMS18] and the references within).

The second is *high-dimensional expanders*. A line of work, started independently by Linial and Meshulam [LM06] and by Gromov [Gro10], defines and studies expansion beyond graphs, in higher dimensional simplicial complexes. The quest to explicitly construct these objects beautifully connects algebra, geometry, and topology. It has already found connections and applications to such computational areas as property testing and quantum error-correcting codes. An introduction to this rapidly moving field is [Lub14]. More recent constructions and applications appear e.g. in [EK16, DK17, KO18, DHK$^+$19].

8.8   Structure vs. pseudo-randomness

This section only exposes a tip of a growing iceberg, in which *pseudo-randomness* and its interaction with *structure* (both defined to suit the occasion) become a very powerful "meta proof technique" in a diverse number of math and CS areas. One beautiful survey by Tao [Tao06] explains in detail how this technique is present in the sequence of works on arithmetic progressions in the integers: Roth's theorem, Szemerédi's theorem, Szemerédi's regularity lemma, Furstenberg's ergodic theory proof, Gowers' quantitative bounds, and the Green-Tao theorem about progressions in the primes. Further, it elucidates the need for and presence of "structure vs. pseudo-randomness" *dichotomy* theorems for a variety of mathematical objects. Tao gives many more applications in other areas, including number theory, partial differential equations, ergodic theory, and graph theory in Chapter 3.1 of [Tao08]. Yet other computational sources of a variety of dichotomy theorems are the attempts (mentioned in Section 8.5) to design pseudo-random generators against weak computational models, some of which we will mention below.

Let us start with one general set-up, which can be specialized to many of the examples discussed above. Let X be a finite set, and let our universe U be all bounded functions on X; specifically, all functions $f : X \to [-1, 1]$. For example, when U is all graphs on n vertices, then X will be the set of all pairs $i \neq j \in [n]$, and a graph G is represented by such a function f as follows: $f(i, j) = 1$ if (i, j) is an edge of G, and $f(i, j) = -1$ otherwise. Note that allowing range $[-1, 1]$ actually allows us to also consider "edge-weighted graphs," or equivalently, convex combinations of graphs.

For any two functions $f, g \in U$, define their *correlation* simply as $\langle f, g \rangle = \mathbb{E}_{x \in X}[f(x)g(x)]$ when the underlying distribution on X is uniform. We will use correlation to define pseudo-randomness. A pseudo-

random property will be defined by a family of test functions $\mathcal{F}$ in U as follows. Pseudo-random functions will be those that are almost orthogonal to every test function in $\mathcal{F}$. More precisely, call a function $g \in U$ "$(\epsilon, \mathcal{F})$-pseudo-random," if for every $f \in \mathcal{F}$, $|\langle f, g \rangle| \leq \epsilon$. This mechanism of defining pseudo-randomness is very general. Indeed, most of the pseudo-random properties listed in previous sections can be expressed in this way. For example, property S_1 of expanders in Section 8.7 is obtained by taking, for every subset T of vertices, the indicator function on the pairs of vertices between T and its complement, and subtracting from it the expected value for a set of that size in a random graph. Thus, in this example, every set T yields a test function.

The following theorem is a template of one basic type of desired *dichotomy* theorem between structure (or "simplicity") and pseudo-randomness.

Theorem 8.11 (Template dichotomy theorem)**.** *Let U and $\mathcal{F}$ be as above. Then every function $g \in U$ can be decomposed as*

$$g = s + e,$$

where s is a "simple" function, and e (for "error") is a pseudo-random function, both with respect to F.

More precisely, there is a real function m such that for every $\epsilon > 0$, there is such decomposition with the following properties. First, the function e will be $(\epsilon, \mathcal{F})$-pseudo-random. Second, the function s is composed from at most a finite number $m(\epsilon)$, depending only on ϵ, of functions from $\mathcal{F}$. Namely, $s = h(f_1, f_2, \ldots, f_m)$ with $m \leq m(\epsilon)$, $f_i \in \mathcal{F}$, and h is some (combining) function.

Let us see how such a dichotomy theorem may be useful for proving statements about *all* objects in U. Basically, it allows us to treat separately simple objects and pseudo-random ones (which are simple in a different, statistical sense). This sounds naive, and indeed, we describe it in a naive, high-level fashion, but it should give a sense of the way some powerful theorems mentioned above are proved. So, suppose you want to prove some universal statement about U, namely, that *every* object in it has some desired property. For example, you may be interested in proving Szemerédi's theorem: For every fixed $\delta > 0$, and every integer k, *every* subset of the first n integers (for large enough n in terms of δ, k) of measure δ must contain a k-term arithmetic progression. In this example U_n is the family of all δ-dense subsets of $[n]$.

The first step is to understand why a *random* object from U satisfies the desired property. Finding sufficient conditions for this[21] may suggest pseudo-randomness "test functions" $\mathcal{F}$, which, in turn suggest, by the dichotomy theorem, what simple "structure" is. In the arithmetic progressions example, it is easy to see that a random subset of $[n]$ of measure δ will have in expectation plenty of k-term progressions, roughly $\delta^k n^2$ of them. So, a chosen pseudo-random property can naturally try to enforce these statistics. Roth's theorem [Rot53] on 3-term arithmetic progressions in dense subsets of integers, which inspired this whole development, does just that. Roth observes that the statistics of such progressions hold if the subset has small correlation with any periodic function, and so he takes the characters of $\mathbb{Z}_n$ to be his family of pseudo-randomness tests. For larger k, Gowers [Gow01] invented his *Gowers norms*, pseudo-randomness tests that similarly enforce the statistics of k-term progressions in random subsets. Of course, the hard part is making the right choices of pseudo-random test functions—to balance between the structured and random-looking parts—in a way that allows proving that each (and their sums) has the desired property.

Let us return to the dichotomy theorems themselves, and try to understand when we can prove such theorems. First, let us spell out a very basic one, indeed, using linear characters. It is extremely simple and can be thought of as a college math homework problem about discrete Fourier transforms. Let $X = \mathbb{F}_2^n$, and U be the set of all functions on X. The dual group $\hat{X}$ has 2^n characters, χ_T, one for every subset T of $[n]$. We will take this collection to be our set of test functions, namely, $\mathcal{F} = \hat{X}$. Giving a formal structure vs. pseudo-randomness theorem, in the template of Theorem 8.11, is easy in this setting. Recall that the functions in $\mathcal{F}$ form an orthonormal basis for $\mathbb{R}^{2^n}$. Thus, every function $g \in U$ has a unique representation in this basis as $g = \sum_T c_T \chi_T$, where the coefficients c_T are called the *Fourier coefficients* of g and are computed by $c_T = \langle \chi_T, g \rangle$. Now the decomposition suggests itself. Given $\epsilon > 0$, call T *large* if $|c_T| \geq \epsilon$ and *small* otherwise, and define the simple part to be $s = \sum_{T \text{ large}} c_T \chi_T$ and the pseudo-random part to be

---

[21]Which is very similar to attempts to de-randomize particular probabilistic algorithms, where we seek sufficient conditions on properties of the random input which will cause the algorithm to give the correct answer.

$e = \sum_{T \text{ small}} c_T \chi_T$. Clearly, $g = s + e$. The function e is $(\epsilon, \mathcal{F})$ pseudo-random by the definition of *small* and the orthogonality of characters. The simplicity of s is argued as follows. The norm $\langle g, g \rangle$ of g is 1, and so Parseval's identity implies that $\sum_{T \text{ large}} (c_T)^2 \leq 1$. As each $|c_T|$ is at least ϵ, there can be no more of $m(\epsilon) = \epsilon^{-2}$ functions in the simple part. Note that the combining function h in s here is extremely simple and efficient (namely, a linear combination).

It should be clear that there was nothing special about the Fourier characters here—all we used was the *orthogonality* of the functions in $\mathcal{F}$. In other words, the template dichotomy theorem holds when $\mathcal{F}$ is an orthonormal basis of U. This seems, and indeed is, an extremely simple case. To what generality can we expect such a dichotomy theorem to hold? Well, in *full* generality! Remarkably, *every* choice of X and $\mathcal{F}$ affords such a decomposition! Many special cases of it appeared, including in Szemerédi's regularity lemma, the Green-Tao work on arithmetic progressions in the primes, and in general (but in different form) in [TZ08, RTTV08]. The following version, which gives the best parameters, is due to Trevisan, Tulsiani, and Vadhan [TTV09].

Theorem 8.12 [TTV09]. *The template dichotomy theorem holds for* every *choice of X and $\mathcal{F}$. Moreover, the bound $m(\epsilon) = O(\epsilon^{-2})$, and the combining function h uses at most ϵ^{-2} simple operations: addition, multiplication, and threshold.*

The proof is essentially greedy, and goes roughly as follows. One constructs the (simple) function s approximating the given g in stages, starting from the constant zero function. If the current $g - s$ has correlation below ϵ with *all* functions in $\mathcal{F}$, we are done. If not, and $g - s$ does have correlation at least ϵ with some member $f \in \mathcal{F}$, then we add to s (an appropriate) constant multiple of f. Finally, a simple potential function is used to bound the number of iterations. This powerful idea and its variants have found uses (often under the names "boosting" or "multiplicative-weight updates") beyond pseudo-randomness in numerous algorithmic and other application areas. We discuss two such (related) applications later in the book, for on-line predictions in Chapter 16 and for amplifying the quality of learning algorithms in Chapter 17. An excellent survey on this meta-algorithm is [AHK12].

Let us demonstrate one (indirect) application of this dichotomy theorem to computational pseudo-randomness. *Every* Boolean function can be "approximated" by an easily computable one, in the sense that their symmetric difference is computationally pseudo-random.

Corollary 8.13. *Let U to be the set of all Boolean functions on n bits. Also, for some fixed c, let $\epsilon = n^{-c}$ and let $\mathcal{F}$ be the set of all n^c-size circuits on n input bits.*

For every Boolean function g, we have $g = s \oplus e$, where $s \in \mathcal{P}/\text{poly}$, and e is computationally pseudo-random—no function in $\mathcal{F}$ has correlation $\geq \epsilon$ with e.

Special cases of this general dichotomy theorem, proved in a similar fashion earlier, were actually motivating sources for it, which arose while studying a variety of objects with different motivations, including the "weak regularity lemma" of Frieze and Kannan [FK96] in graph theory, the "dense model theorem" of Tao and Ziegler [TZ08] used for progressions in primes, and the "hard-core set" theorem of Impagliazzo [Imp95a] from computational pseudo-randomness. Trevisan, Tulsiani, and Vadhan's paper [TTV09] gives three quite different proofs representing different origins; one using the "boosting" technique from computational learning theory, one using the minimax theorem from game theory, and one using the recursive refinement arguments à la Szemerédi.

As mentioned, there is a great variety of dichotomy (or "decomposition") theorems between randomness and structure, in other settings, which may differ in form but have the same essence. Some of the recent mathematical objects for which such results were proved are listed below. Some are quite a bit more complex than the template above, but in some cases, like the first item below, the situation is even simpler, in that every object itself is either structured or pseudo-random:

- Bounded degree polynomials over finite fields [GT09, KL08].

- Bounded degree polynomials in Gaussian variables [Kan12, DS13].

- Bounded sensitivity Boolean functions [Hat10].

- Bounded degree polynomial threshold functions [DSTW14].

- Hypergraphs [RS06].

- Inverse theorem for the Gowers' norms [GTZ12, Sze12].

9 Weak random sources and randomness extractors

Probabilistic algorithms and many other applications of randomness are analyzed assuming access to an unlimited supply of *independent, unbiased* bits. Does reality provide such perfect randomness? Suppose nature is deterministic, and perfect randomness is simply nonexistent. Even then, Section 7.2 demonstrates that believable *hardness assumptions* imply $\mathcal{BPP} = \mathcal{P}$ (namely, every probabilistic algorithm can be efficiently de-randomized), and so all these algorithmic applications of perfect randomness survive in a deterministic world. But suppose that we want *unconditional* results. What are the minimal assumptions about nature that will afford the same algorithmic applications?

A reasonable middle ground regarding nature, which seems to be supported by experience, is that even if it does not provide us with perfect random bits, many of its processes are, to some extent, *unpredictable*. This includes the weather, stock markets, Internet traffic, sunspots, radioactive decay, quantum effects, and a host of others, which we can tap into for randomness. Indeed, in practice, many computer systems generate the random bits needed for a variety of algorithms in precisely this way. Sampling such processes generates a stream of possibly *correlated, biased* random bits.[1] This leads to obvious questions. What is a good mathematical model for such *weak* randomness? Can we use it for applications requiring perfect randomness? How?

Three decades of study have generated a beautiful theory, answering these questions and others. These developments are surveyed in detail in [Nis96, Sha04, Vad11], and I briefly summarize them below. First we discuss a formal mathematical model of weak random sources. Then I describe the hero of this section, the *randomness extractor*, a deterministic algorithm whose role is to "purify" a random sample from any weak random source into a perfect (or near-perfect) sample from a uniform distribution (which in turn is usable in applications). *As it happens, randomness extractors, born for the this purpose, turned out to be useful, even essential, in a variety of diverse (theoretical and practical) application areas, including error-correcting codes, data structures, algorithms, de-randomization, cryptography and quantum computing.*[2] Interestingly, in many of these applications, randomness is completely absent as resource or motivation. Nonetheless, the *pseudo-random* properties of extractors make them applicable to many other desiderata, much like the case of expander graphs in Chapter 8. For all of these applications, we need *efficiently computable* extractors, and I will describe explicit constructions of such efficient extractors. Note that the extractors discussed here are sometimes called *seeded extractors*, to distinguish them from a more restricted cousin, *deterministic extractors*. The latter deals only with restricted families of sources, as does another related area of research— *data compression* [ZL78]. Here we survey only work on the most general class of weak random sources, which have some entropy but otherwise have no structural restrictions.

To get a feel for what weak sources may look like, and the challenge of purification, here are a few examples. To warm up: How would you purify a sequence of *independent* tosses of the same biased coin, without knowing the bias, except that it is in the range, say, $[0.1, 0.9]$? Specifically, find a deterministic algorithm that converts an n-bit sequence from such a source into another (possibly shorter) sequence that is uniformly distributed (try doing this yourself, or peek at the footnote).[3] Here are some other examples of "weak" sources. First, consider a sequence of independent coin tosses as above, but where each toss has a possibly different unknown bias in this range. Next, consider a sequence generated one bit at a time by an adversary, who, depending on past toss outcomes, picks a bias from this range and tosses the next coin with this bias. Finally, consider a sequence of n bits in which an adversary tosses independent, unbiased coins in (say) $n/10$ bit positions of her choice, and then uses the outcomes to determine the values in the remaining bit positions deterministically.

Note that while such distributions may indeed arise from sampling weak natural sources, they can also

---

[1]Note that even though quantum mechanics predicts that the measurements of (say) successive photon spins yield perfectly random bits, the physical devices generating and measuring these will not be perfect.

[2]A comprehensive survey of these amazingly diverse applications of extractors is sadly missing. One can base a fantastic graduate course on the extractor application areas which can be found in the papers [RR99, TSZ04, Vad04, Zuc06, Vad09, Wig09, GZ11, MS16, Li17] and the references therein.

[3]The idea, going back to von Neumann, is simple. Pair up the bits of the input sequence, and consider them left to right. Ignore the pairs 00 and 11. For each pair 01 output, say heads and for each 10 output, call it tails. Note that each output symbol is independent of the others and has probability $\frac{1}{2}$.

arise in practical computing scenarios. For example, if you pick a completely random n-bit key for cryptographic purposes, but parts of it leak to an adversary (possibly a small bias of each bit, or perhaps the values of an unknown subset of $n/10$ of them), the conditional distribution on your key is of the above mentioned types. A completely different scenario leading to similar distributions arises in the construction of pseudo-random generators for space-bounded computation (discussed in Chapter 14). These situations illustrate why the use of randomness extractors exceeded their initial motivation, as mentioned above.

What does it mean to extract pure randomness from such weak sources? The chosen purification method should work for *every* distribution in the class—I used the word "*adversary*" to stress that the distribution is picked *after* the purification method was chosen. Note that all the above examples of probability distributions have entropies[4] that are very substantial: a constant fraction of the length n. However, it is far from clear how to use this fact, as the purifier doesn't know where this entropy is "hiding." It is important to stress that the purifier gets only a *single* n-bit sample from the unknown distribution—neither nature nor adversaries would do us the favor of providing several independent samples from it.

9.1 Min-entropy and randomness extractors

9.1.1 Min-entropy: Formalizing weak random sources

John von Neumann was probably the first to ask the question, in the 1940s, of how to use an imperfect random source. Von Neumann actually needed perfect random bits for Monte Carlo simulations on his "IAS machine" (one of the earliest computers). I explained his solution for the warm-up example above. In the 1980s, starting with [Blu86], came a sequence of different models of weak random sources (and ways of "improving their quality"), importantly [SV86, CG88, VV85]. A complexity-theoretical motivation for studying weak sources was given by Sipser [Sip88]. Finally, Zuckerman [Zuc90] gave the ultimate definition of the distributions we may hope to purify; a weak random source is simply modeled as an *arbitrary* probability distribution, on $\{0,1\}^n$, which has some amount k of entropy in it. It turns out that the right notion of entropy to take is *min-entropy* defined below, as opposed to the classical Shannon entropy.[5] Min-entropy is simply the logarithm of the L_∞ norm of the probability distribution.[6]

Definition 9.1 (*Min-entropy*)**.** Let D be a probability distribution on $\{0,1\}^n$, and let D_x denote the probability of a sequence $x \in \{0,1\}^n$. The *min-entropy* of D, denoted $H_\infty(D)$ is the minimum, over all $x \in \{0,1\}^n$, of $-\log_2 D_x$.

Definition 9.2 (*k-source*)**.** We say that D is a *k-source* if $H_\infty(D) \geq k$, namely, if every sequence x occurs in D with probability at most 2^{-k}.

It is instructive to convince yourself that all four examples of weak sources mentioned above are k-sources for $k = \Omega(n)$. A convenient way to think about a k-source D, which turns out to lose no generality in the sequel, is simply as the uniform distribution over some (unknown) subset $S \subset \{0,1\}^n$, of size at least 2^k.

Of course, we continue to think asymptotically, so while discussing fixed n and k we really consider ensembles of distributions $D = D_n$, and allow the min-entropy $k = k(n)$ to depend on n (e.g., to be $\sqrt{n}$). Moreover, the purification algorithm will have to be efficient in terms of n.

9.1.2 Extractors: Formalizing the purification of randomness

The purification algorithm is called a *randomness extractor*, or briefly, an *extractor*. Let us try a naive formulation of it, observe its flaws, and then fix it to be the correct definition. As extractors must be

---

[4]For now the reader can think about entropy informally as "randomness content" or formally as Shannon's entropy. We will soon define the notion of entropy that is actually relevant to this setting.

[5]The reason for this choice is that there are distributions with extremely high Shannon entropy in which a single sequence has high probability, and this makes randomness extraction impossible.

[6]This notion appeared first in this context of randomness extraction (but was used to define a more restricted class of sources) in [CG88].

deterministic, it is natural to consider a function $f : \{0,1\}^n \to \{0,1\}^r$ an (n,k)-extractor if for *every* k-source D on n bits, $f(D)$ is *statistically close* to U_r, the uniform distribution on r bits. In other words, we have the L_1 distance $|f(D) - U_r|_1$ is at most ϵ, which for this section is best taken to be $\epsilon = 1/\text{poly}(n)$ (even though for some applications a small constant suffices).

Clearly, we must have $r \leq k$, as a deterministic process cannot increase entropy.[7] Unfortunately, such functions f simply do not exist. They fail to exist even in the extreme case where $k = n - 1$ (namely, the entropy is almost everything), and in any event, we are trying to extract only one bit, namely, $r = 1$. The reason is that for the Boolean function f, at least one of $f^{-1}(0)$ or $f^{-1}(1)$ has size at most 2^{n-1}. So, let D be the uniform distribution on the larger set of the two, and note it has min-entropy of at least $n-1$. However, the distribution $f(D)$ is constant and is statistically as far as possible from the uniform distribution on 1 bit.

The right definition of randomness extractors, in the sense that they exist and are still useful (as we'll see) for the purpose of purification, was given by Nisan and Zuckerman [NZ96]. It allows using not one function f but many functions, and demands that most of them will purify any given source.

Definition 9.3 (*Extractor*). A sequence of functions $F = (f_1, f_2, \ldots, f_t)$ with $f_i : \{0,1\}^n \to \{0,1\}^r$ is called an $(n,k) - extractor$, if for every k-source D, all but ϵ-fraction of the f_i satisfy $|f_i(D) - U_r|_1 \leq \epsilon$.

Remark 9.4. In most papers on the subject, the extractor F is defined differently, but essentially equivalently, as a function of two arguments: the sample from the weak source D, as well as a (uniformly distributed) index $i \in [t]$, so that $F(i, x) = f_i(x)$. The index is called a *seed*, and F is often called a *seeded extractor*.

Enhancing weak random sources with extractors   Let us discuss how to use an extractor to emulate $\mathcal{BPP}$ algorithms while only accessing weak random sources with sufficient entropy. Fix some probabilistic algorithm A for some decision problem and an input x for it. Assume that on a truly random sequence $y \in \{0,1\}^r$, its output $A(x,y)$ errs with probability (say) at most $1/5$. Assume further that we have an (n,k)-extractor F as above, with output size r and error ϵ. The emulator would appeal to some k-source D for an n-bit sample z, and compute from it the set of r-bit sequences $y_i = f_i(z)$. Then it would plug each of the y_i into A, computing all outputs $A(x, y_i)$ and then returning their majority value. Let us analyze the quality of this new algorithm. We know that for an ϵ-fraction of the seeds i, the function f_i may fail to produce a nearly uniformly random (up to ϵ) sequence when applied to a sample from D. These seeds can produce an erroneous value. But each of the other seeds generates a nearly uniform sequence y_i, and so when used in A, errs with probability at most $1/5 + \epsilon$. By Markov's inequality, the probability that a total of at least half the seeds lead to an error is therefore at most $2/5 + 4\epsilon < 1/2$.

Note that for such an emulation to be efficient (namely, polynomial time in r) puts some restrictions on the extractor F. First, we must have $t \leq r^{O(1)}$. Next, each f_i should be computable in $\text{poly}(r)$ time. Thus in particular, we must have that $n \leq r^{O(1)}$. Finally, as $k \leq r$, this implies we can only hope to use sources with polynomial entropy $k \geq n^{\Omega(1)}$.

Important parameters of extractors   As mentioned, the above application of using weak random sources in probabilistic algorithms was the first, but far from the last application of extractors throughout theoretical and practical computer science. Later applications of randomness extractors are interesting for different parameters than the ones above, so let us consider *all* the possible parameters in this definition of extractors and the natural goals in optimizing them. We will express all parameters as functions of n, the sample size from the distribution D. First, the min-entropy k; it would be nice to extract from sources of any k (although it seems clear that the larger k is, the easier the task becomes). Next is the output length r; how many (nearly) pure random bits are produced. As mentioned, $r \leq k$, and it is natural to make it as close to k as possible. Next is the "error" parameter ϵ, which we would like to make as small as possible. Finally, the number t of functions used; again, this should be minimized, as we have to evaluate them all (for efficiency, a natural goal is to make t polynomial in n).

---

[7]This is a well-known fact for Shannon's entropy and is actually easy to see for min-entropy as well.

Early results [Sip88, RTS00] showed that at least *existentially*, one can simultaneously get the best of all worlds. Namely, one can get optimal values for all parameters: output entropy r being almost equal to the input entropy k, a polynomial number of functions t, and inverse polynomial error ϵ. Of course, such existential results do not give bounds on the efficiency of the extractor but instead clarify the limits of what we should aim for in efficient constructions.

Theorem 9.5. *For every n and $k \leq n$, there* exists *an (n,k)-extractor F with output length $r \geq .99k$, $\epsilon \leq 1/n$, and $t = n^{O(1)}$. Indeed, a* random *family of t functions F will be such an extractor with probability approaching 1 as n grows. Furthermore, these parameters are essentially the best possible.*

9.2   Explicit constructions of extractors

Of course, the main issue is that we need to actually use extractors, so an existence theorem will not do—we need an explicit construction of *efficiently computable* extractors F with good parameters.[8] The road to achieving this goal, detailed in the surveys cited above, was long and meandering, involving boosting from weaker classes of sources (like "block-sources" and "somewhere-random sources") and using a variety of weaker notions of extractors (like "condensers" and "mergers"). All these constructs gave rise to new and different notions of reduction. I now highlight only a few milestones of explicit extractors that gradually lowered the entropy requirements of the source, from linear, to polynomial, to anything at all. I also highlight the diverse intellectual and technical origins that gave rise to different constructions, which further points to the interconnectedness of the pseudo-random world. In the following discussion, assume for simplicity $\epsilon = 1/n$. We only focus on minimizing the entropy k of the source,[9] and the number of functions (or seeds) t, and maximizing the output length r of the source (which cannot exceed k).

At the end of this historical account, I give one explicit construction of an extractor, and another application of extractors in which weak sources are not mentioned.

The very first explicit extractor was given by Zuckerman [Zuc90, Zuc91]. It could only handle entropy $k \geq \Omega(n)$ and outputs of $r \geq k/10$ nearly uniform bits. It involves a sophisticated combination of randomness-efficient sampling and hashing. After a few improvements, Ta-Shma introduced and used *mergers* to make a huge leap. He showed how to extract from a source of any min-entropy $k \geq \text{poly}(\log n)$, output length r essentially as large as k, with only a slightly super-polynomial number of seeds $t = \exp(\text{poly}(\log n))$. The quest became to reduce the number of seeds.

The next milestone is the explicit extractor of Trevisan [Tre99], which achieved the optimal $t = \text{poly}(n)$ but only for polynomial entropy $k = n^{\Omega(1)}$ and only with output length $r > k^{.99}$. This sufficed to completely resolve the $\mathcal{BPP}$ emulation problem by weak sources discussed above. Moreover, this construction was conceptually very different than all previous constructions. Indeed, it was a reduction. Trevisan's extractor interprets the input as a truth table of a *computationally hard function g*, and the functions f_i output values of g on judicially chosen domain elements. The construction and analysis follow the NW-generator constructions [NW94, IW97] mentioned in Section 7.2. Let me remark on how insightful and surprising this construction was. First, it uses an object (pseudo-random generator) that by definition works only in the computational setting and converts it to another object (extractor) that by definition is information-theoretic. Moreover, these two types of notions seem to work in opposite directions: Pseudo-random generators start with few truly random bits and generate a low-entropy distribution on many bits, whereas the extractors start with a distribution on many bits that has some entropy and generate a few, purely random bits. Nevertheless, Trevisan shows that the NW-generator, essentially as is, becomes an extractor when viewed from the right perspective!

This story continued to evolve with many ideas and papers, and finally reached a happy ending: efficient extractors that are essentially optimal in all parameters.

Theorem 9.6. *For every $k = k(n)$, there is a polynomial-time computable family $F = \{F_n\}$ of (n,k)-extractors, with output length $r \geq .99k$, $\epsilon \leq 1/n$, and $t = n^{O(1)}$.*

---

[8]Note that this is another instance of the "hay in a haystack" problem discussed in Chapter 8.

[9]Which we assume for notational simplicity is at least $10 \log n$.

The first such explicit construction was given by Guruswami, Umans, and Vadhan [GUV09]. Their extractor interprets the input as a *message in an error-correcting code*, and the functions f_i output the different symbols in the encoding of that message. The construction and analysis relies specifically on the optimal list-decodable codes of [PV05, GR08b].

Shortly after, a different construction was given by Dvir and Wigderson [DW11]. Their extractor interprets the input as a *low-degree curve over a finite field*, and the functions f_i output the different points on the curve. The construction and analysis rely on the *polynomial method* and its use in Dvir's proof [Dvi09] of the finite-field Kakeya conjecture in finite-field geometry.

The last two results draw on and connect to different mathematical areas, an aspect shared by many other works on extractors, especially in their numerous application areas mentioned in the beginning of this chapter.

Extractors from expanders We conclude with an explicit construction of an extractor, from yet a different origin: random walks on expander graphs. Indeed, this may be called the protoextractor, as it existed before extractors were defined but was realized to be an extractor only later. Moreover, it was used as part of many subsequent extractor constructions. This extractor has relatively weak parameters; for some fixed constant α, the entropy of the source k must be at least $(1 - \alpha)n$, the output length is $r \geq \alpha k$ and a constant error ϵ. Still, even obtaining this (without hindsight) is highly nontrivial.

We shall need the explicit expanders of Theorem 8.9, say, with parameters $c = 2$ and $d = 16$. Let G_r be the 16-regular expander on 2^r from that theorem.[10] Set $t = r/\epsilon^2$ and $n = r + 4(t-1)$. Note that every n-bit sequence can be interpreted as a length-t path in G_r, where the first r bits specify the first vertex, and each successive 4-bit segment specifies a neighbor of the previous vertex among the 16 possible ones. Note that $r \approx \epsilon^2 n/4$.

Define the t functions $F = (f_1, f_2, \ldots, f_t)$ with $f_i : \{0, 1\}^n \to \{0, 1\}^r$ as follows: $f_i(x)$ is simply the (r-bit name of the) ith vertex in the path specified by x. By Theorem 8.9, F can be computed in polynomial time.

Theorem 9.7. *Set $\alpha = (\epsilon^2 t)/(32n) = \Omega(\epsilon^2)$. Then F is an explicit (n, k)-extractor for $k = (1-\alpha)n$, $r \geq \alpha k$, and error ϵ.*

The proof of this theorem follows from a remarkable sampling property of random paths in expander graphs: Their t vertices, despite being a highly correlated set of r-bit strings, behave as totally independent ones when used to compute a sample average of any bounded function on $\{0, 1\}^r$; the deviation from the true average decays exponentially with the number of samples t. This was first discovered (in a weaker form) by Ajtai, Komlós, and Szemerédi [AKS87] (for the purpose of de-randomizing small-space probabilistic algorithms). It was strengthened in [CW89, IZ89] for the purpose of amplifying error in probabilistic algorithms, and finally, Gillman [Gil98] proved the essentially optimal bound (simplified and extended in [Hea08], from which we quote a special case in our notation, with his $\lambda = 1/2$).

Theorem 9.8 [Gil98, Hea08]. *Let G_r be the expander above. Fix any function $g : \{0, 1\}^r \to [-1, 1]$ with zero expectation. Then for every $\epsilon > 0$ and every t, if $y_1, y_2, \ldots, y_t$ are the vertices of a uniformly random path in G_r, then*

$$\Pr\left[\left|\sum_i g(y_i)\right| > t\epsilon\right] < 2^{-\epsilon^2 t/8}.$$

Observe that when the y_i are independent, this is the classical large-deviation (Bernstein/Chernoff) bound. The huge difference is that independent samples require rt random bits, whereas this theorem shows that the same estimation error can be achieved with only $r + O(t)$ bits, which is the best possible. The application to error amplification of probabilistic algorithms mentioned above shows that any algorithm using r random bits with error (say) $1/3$ can be converted into one with error $\exp(-r)$ by using only $O(r)$ bits, as opposed to the obvious r^2. This may be viewed as a far more impressive result than the error reduction[11] discussed in Section 8.7.

---

[10] Note that we change n of that theorem to r here, as we reserve n for the input length of the extractor.

[11] There the error was reduced to $1/\text{poly}(r)$, but without adding any extra random bits at all.

We conclude by connecting the last two theorems. They turn out to be essentially equivalent (once you match the parameters). This equivalence of extractors and (oblivious) samplers is stated by Zuckerman in [Zuc97], and the simple proof is almost by definition. Vadhan's survey [Vad11] contains a thorough discussion of the connections of extractors to samplers, hash functions, error-correcting codes, and other pseudo-random objects.

9.3   Structured weak sources, and deterministic extractors

This chapter has focused on *seeded extractors*, which take a few extra truly random bits to purify weak random sources. As explained, this is necessary for the lofty goal they perform, namely purifying *arbitrary* weak random sources of a given min-entropy, putting no structural requirements on these sources. Indeed, it is this ability that gives seeded extractors their amazingly diverse applicability.

But in many cases we know, or can hypothesize, additional structure of the weak sources; in a sense restricting the form in which entropy is scattered across a sequence drawn from the distribution defined by the source. For such (families of) sources, can one construct completely deterministic extractors (without an extra random seed) to purify them? The answer is positive is many different cases, which we briefly enumerate, and provide some key references to start off the interested reader. Needless to say, deterministic extractors have diverse applications as well, and their constructions beautifully interact with other areas of math and CS (and in particular, with constructions of seeded extractors).

The types of structured random sources that were studied and for which deterministic extractors were constructed fall roughly into three classes. The first is *algebraic*, where the distribution is generated by natural algebraic functions. These include "affine sources" [GR08a, Yeh11, Li16], "polynomial sources" [DGW09] and "varieties" [Rem16]. The second is *computational*, where the distribution is generated by functions with bounded resource (time, space, or other) computations [TV00, KRVZ11]. The third is *independent sources*, where the distribution samples *several* unstructured sources [CG88, BIW06, CZ16] (the last paper is the breakthrough on explicit Ramsey graphs mentioned in Section 8.2).

It is a very interesting research direction to try and classify the structured sources which admit deterministic extractors. An important definition, capturing many of the above structures, called *sumset sources*, was given by Chattopadhyay and Li [CL16].

10 Randomness and interaction in proofs

The introduction of randomness, and interaction, into *proofs* (and proof systems), completely rejuvenated this central tenet of mathematics. It had a remarkable impact on theoretical computer science and many related areas, with quite a number of unexpected consequences. In particular, it resulted in new, powerful characterizations of $\mathcal{NP}$ and other complexity classes.

In this chapter, we survey the main definitions and results on probabilistic and interactive proofs, and their meaning and impact. The monograph [Gol99] offers much more detail and precision; a comprehensive survey of how this field evolved in the past 2 decades would be welcome! We note that in this section, we do *not* discuss the *probabilistic method*, a powerful proof *technique*. An excellent text on it is [AS00]. Finally, the reader is encouraged to recall the meaning of $\mathcal{NP}$ as a proof system from Chapter 1.5, and the notion of propositional proof systems from Chapter 15.2.4, which will be extended here using randomness and interaction.

Let us start with an example. Consider the graph isomorphism problem mentioned in Chapter 3.3: given two graphs G and H, determine whether they are isomorphic. No polynomial-time algorithm is known for this problem.[1] Now assume that an infinitely powerful teacher (who in particular can solve such problems), wants to convince a limited, polynomial-time student, that two graphs G, H are isomorphic. This is easy— the teacher simply provides the bijection between the vertices of the two graphs, and the student can verify that edges are preserved. This is merely a rephrasing of the fact that *ISO*, the set of all isomorphic pairs (G, H), is in $\mathcal{NP}$. But is there a similar way for the teacher to convince the student that two given graphs are *not* isomorphic? It is not known if $ISO \in \text{co}\mathcal{NP}$, so we have no such short certificates for nonisomorphism. What can be done?

Here is an idea from [GMW91], which allows the student and teacher more elaborate interaction, as well as coin tossing. The student challenges the teacher as follows. She (secretly) flips a coin to choose one of the two input graphs G or H. She then creates a *random* isomorphic copy K of the selected graph, by randomly permuting the vertex names (again with secret coin tosses). She then presents the teacher with K, who is challenged to determine whether K was generated from G or H. Observe that if G and H are indeed nonisomorphic, as claimed, then the answer is unique, and the challenge can always be met by the teacher (who has infinite computational power). If, however, G and H are isomorphic, *no* teacher can guess the origin of K with probability greater than $1/2$. Simply put, the two random variables—a random isomorphic copy of G and a random isomorphic copy of H—are *identically distributed*, and so cannot be told apart, regardless of computational power. Now, to reduce the error, let the student repeat this experiment independently 100 times, and declare that the graphs are not isomorphic unless the teacher succeeds in all of them. As the probability of 100 successes when G and H are isomorphic is 2^{-100}, this bounds the probability that the student erroneously accepts a false proof. In other words, such repeated success describes an overwhelmingly convincing *interactive proof* that the graphs are indeed nonisomorphic.

Note that hiding the coin tosses of the student from the teacher is an absolutely essential feature of this proof system. Indeed, it is hard to imagine that a similar feat can be achieved if the teacher could spy over the student's shoulder and know precisely the results of all coin tosses used. This intuition is wrong! A remarkable result of Goldwasser and Sipser [GS89] gives another (much more sophisticated) interactive proof system for nonisomorphism, in which all coin tosses of the student are available to the teacher. Indeed, they give a completely general way of turning any "private-coin" interactive proof system (in which the teacher can't see the student's coin tosses) into one that is "public-coin" (in which the teacher can see them), which has similar efficiency. The reader is encouraged to try to find an interactive proof for graph nonisomorphism, in which the only messages of the verifier to the prover are random bits.[2]

Let us now return to the general discussion. We have already discussed proof systems in Section 3.3 and Chapter 6. In both, the *verifier* that a given witness to a given claim is indeed a proof was required to be an efficient *deterministic* procedure. In the spirit of the previous chapters on randomness we now relax this requirement and allow the verifier to toss coins and err with a tiny probability.

---

[1] Although the recent breakthrough of Babai [Bab15] (see also the exposition in [HBD17]) comes quite close!

[2] Insufficient hint: The public-coin proof, like the private-coin proof above, should rely on the fact that the number of isomorphic copies of G and H together is twice as large when they are nonisomorphic than when they are isomorphic.

To make the quantifiers in this definition clear, as well as to allow more general interactions between the prover and the verifier, it will be convenient to view a proof system for a set S (e.g., of satisfiable formulas) as a *game* between an all-powerful prover and the (efficient, probabilistic) verifier: Both receive an input x, and the prover attempts to convince the verifier that $x \in S$. Completeness dictates that the prover succeeds for every $x \in S$. Soundness dictates that *every* prover fails for every $x \notin S$. In the definition of $\mathcal{NP}$, both these conditions should hold *with probability* 1 (in which case, we can think of the verifier as deterministic). In probabilistic proof systems, we relax this condition and only require that soundness and completeness hold with high probability (e.g., 2/3, as again the error can be reduced arbitrarily via iteration and majority vote). In other words, for every input, the verifier will only rarely toss coins that will cause it to mistake the truth of the assertion.

This extension of standard $\mathcal{NP}$ proofs was suggested independently in two papers—one by Goldwasser, Micali, and Rackoff [GMR89] (whose motivation came from cryptography, in which interactions of this sort are prevalent) and the other by Babai [Bab85] (whose motivation was to provide such interactive "certificates" for natural problems in group theory, which were not known to be in co$\mathcal{NP}$). While the original definitions differed (in whether the coin tosses of the verifier are known to the prover or not), the paper of Goldwasser and Sipser [GS89] mentioned above showed both models to be equivalent.

This relaxation of proofs is not suggested as a substitute for the notion of mathematical truth. Instead, much like probabilistic algorithms, it is suggested to greatly increase the set of claims that can be efficiently proved in cases where tiny[3] errors are immaterial. As we shall see below, *probabilistic proof systems yield enormous advances in computer science, while challenging our basic intuitions about the very nature of proof.* We exhibit three different remarkable manifestations of this statement:

- Many more theorems can be efficiently proved.

- Every theorem can be proved without revealing *anything* about the proof except its validity.

- Every theorem possesses written proofs that verifiers can check by inspecting only a handful of bits.

10.1 Interactive proof systems

When the verifier is deterministic, interaction does not add power, as the prover can predict all future questions. Thus, in this case, we can always assume that the prover simply sends a single message (the purported "proof"), and based on this message, the verifier decides whether to accept or reject the common input x as a member of the target set S. In other words, with a deterministic verifier, interactive proofs can only prove statements in $\mathcal{NP}$.

When the verifier is probabilistic, *interaction* may add power. We thus allow both parties to toss coins, and consider a (randomized) interaction between them. It may be viewed as an "interrogation" by a persistent student, asking the teacher a series of "tough" questions in succession in order to be convinced of correctness (or to catch a bug). Since the verifier ought to be efficient (i.e., run in time polynomial in $|x|$), the number of such rounds of questions is bounded by a polynomial.[4]

Definition 10.1 (The class $\mathcal{IP}$ [GMR89, Bab85])**.** The class $\mathcal{IP}$ (short for "interactive proofs") contains all sets S for which there is a probabilistic polynomial-time verifier that accepts every $x \in S$ with probability 1 (after interacting with *some* adequate prover) but rejects any $x \notin S$ with probability at least 1/2 (no matter what strategy is employed by the prover, or how computationally strong the strategy is).

We have already seen the potential power of such proofs in the example of graph nonisomorphism above (in the introduction to this chapter), and several other examples were initially given. But the full power

---

[3]And we remind the reader that the error can be made exponentially tiny without affecting efficiency by much!

[4]Restricting the number of rounds to a constant, as was suggested in the original paper of Babai [Bab85], leads to the extremely interesting "Arthur-Merlin" complexity classes $\mathcal{AM}$ and $\mathcal{MA}$, which sit just above $\mathcal{NP}$. We will not define and study them here but note that they were extensively studied, and that the interactive proof above for graph isomorphism above puts this problem in the class $\mathcal{AM}$.

of $\mathcal{IP}$ began to unfold only after an even stronger proof system called "$\mathcal{MIP}$" was suggested by Ben-Or et al. [BOGKW89] (motivated by cryptographic considerations!). In $\mathcal{MIP}$ (short for "multiple-prover interactive proof"), the verifier interacts with *multiple provers*, who are not allowed to communicate with one another. We describe some of the evolution of works and ideas leading to this understanding. A lively account of this rapid progress is given by Babai [Bab90] (see also [O'D05]).

One milestone, by Lund, Fortnow, Karloff, and Nisan [LFKN90], was showing that $\mathcal{IP}$ proofs can be given for *every* set in co$\mathcal{NP}$ (indeed, for much more, but for classes we have not defined in this book). Thus, in particular, *tautologies* have short interactive proofs. Recall that we don't expect these to have standard $\mathcal{NP}$-proofs, as this would imply $\mathcal{NP} = \text{co}\mathcal{NP}$ (see the discussion in Section 3.5, and in particular, Conjecture 3.8).

Theorem 10.2 [LFKN90]. co$\mathcal{NP} \subseteq \mathcal{IP}$.

As mentioned, this theorem is just a special case of the main theorem in [LFKN90], which I state and sketch at the end of this section.

This paper was shortly followed by a complete characterization of $\mathcal{IP}$ by Shamir [Sha92]. He proved its equivalent to $\mathcal{PSPACE}$, the class of functions computable with polynomial memory (and possibly exponential time). Note that this class contains problems that seem much harder than $\mathcal{NP}$ and co$\mathcal{NP}$ (e.g., finding optimal strategies of games).

Theorem 10.3 [Sha92]. $\mathcal{IP} = \mathcal{PSPACE}$.

It is illuminating to give an informal consequence of this theorem, which I find mind-blowing. Suppose that some superior extraterrestrial being (let us call it "E.T." for short) arrived on Earth and claimed that their civilization has studied chess and has found that *"White has a winning strategy"*.[5] Is there a way to check or refute this claim? While we have some very good chess players and programs, our own civilization has no means today of ascertaining such a claim. Simply, the only known way (algorithm) we have is brute force; expanding the complete game-tree for chess. But this tree is exponential in the number of moves, a vast number that puts this computation beyond any conceivable future technology. Of course, we can offer our best players (human or not) to compete playing Black with E.T. But suppose they all lose in all games; all we can conclude is that E.T. is a better player, something far short of verifying its claim that White has a winning strategy. However, Theorem 10.3 does provide an efficient way to verify it!

To explain this subtle connection, consider first a ridiculously stupid attempt at verification. Let us match E.T. not with our best player, but with our dumbest; one who picks each move completely at random, from all available legal moves at the given configuration. Call this probabilistic strategy (for Black) *Random Play* (or R.P. for short). Needless to say, even a beginner playing as White would beat R.P. at chess, and E.T. certainly will, too. The next observation is that R.P., stupid as it may seem, *is* optimal for some other games (e.g., rock-paper-scissors).[6] But of course, rock-paper-scissors has nothing to do with chess.

Amazingly, Shamir's theorem allows us to make the random-play strategy useful in games intimately related to chess! The theorem provides a new kind of reduction: a way to convert chess to a new 2-player game, G, with the following properties.

1. First, the conversion is efficient: The rules of the game G are easily understood by mortals like us. In particular, which moves are legal in each configuration of G are easily computable, and so R.P. in this game can be implemented easily.

2. Second, the two games are equivalent in the following sense. White has a winning strategy in the new game G *iff* White has a winning strategy in chess. So, in particular, E.T. can convert its claimed winning chess strategy (if one exists) into a winning strategy for G.

---

[5] Recall that a *strategy* for a player in chess, or any perfect-information game, is simply a prescription of a legal move for that player in every possible configuration of the game. It is a *winning strategy* if it guarantees a win for that player, regardless what strategy the opponent chooses.

[6] Ignore the fact that this game is of a somewhat different type of game, in which moves are simultaneous. There are other examples. Indeed, the result of Goldwasser and Sipser [GS89] mentioned in the previous Section 10 reveals surprising power of random play in a related context.

3. Finally, G has the property that random play R.P. for Black is *nearly optimal*—it does as well as the best Black strategy with probability $1/2$.

So, let us spell out what this reduction implies. If White has a winning strategy in chess and E.T. uses it to play optimally in G, as it can by property (2), then R.P. would still lose every time in G. But if White does not have a winning strategy in chess, then by property (3), E.T. cannot win G with higher than $1/2$ probability against an optimal Black strategy. Thus, if E.T. wins 100 games in a row against R.P., we have probability $1 - 2^{-100}$ that White indeed has a winning strategy in chess.

Three comments may answer some questions you may have about this result. First, the same reasoning would work for go instead of chess, and indeed, any reasonable game we play. Second, if we appropriately generalize chess or go (and make it a complete problem for the class $\mathcal{PSPACE}$, which happens to be the natural home of such 2-player perfect-information games), then this game-theoretic interpretation is actually equivalent to Theorem 10.3. Third, if $\mathcal{P} = \mathcal{PSPACE}$ (which would imply $\mathcal{P} = \mathcal{NP}$, which no one believes but is still a mathematical possibility), then there is a fast algorithm to determine optimal strategies in chess, go, and similar games.

This success story, of completely understanding the surprising power of interactive proofs, required the confluence and integration of ideas from different "corners" of computational complexity, again exposing the power of its methodology and the flow of cross-fertilizing ideas across different subareas. The sequence of results leading to the original proof uses the following ingredients (elaborating on these is unfortunately beyond our scope):

- *Program checking and testing* [BK89, BLR93].

- *Hardness amplification* from average-case to worst-case hardness [Lip91, BF90].

- The power of counting and the permanent polynomial to capture co$\mathcal{NP}$ and more generally, bounded alternation [Tod91].

- The multi-prover analog [BOGKW89] of the basic interactive proof model, motivated by zero-knowledge proofs (see below).

- The structure of the complexity class $\mathcal{PSPACE}$. In particular, that nondeterminism does not add power to this class [Sav70], and the structure of a specific complete problem for it (namely *QBF*: the satisfiability of quantified Boolean formulas).

A central technical tool that emerged and plays a major role in Section 10.3 (as well as in new lower-bound proofs), is *arithmetization*, the arithmetic encoding of Boolean formulas by polynomials and the ultrafast verification of their properties. Arithmetization and its consequences proved instrumental to some circuit lower bounds; its impact and limitations are mentioned toward the end of Section 5.1.

I conclude by noting that the exact power of the stronger $\mathcal{MIP}$ proof system was also determined completely, by Babai, Fortnow, and Lund [BFL91]. Here, too, it is equivalent to a natural complexity class, in this case $\mathcal{NEXP}$ (the exponential-time analog of $\mathcal{NP}$) of all languages computed in nondeterministic exponential time.

Theorem 10.4 [BFL91]. $\mathcal{MIP} = \mathcal{NEXP}$.

10.2   Zero-knowledge proof systems

This section contains perhaps the most counterintuitive mathematical result you will ever meet!

Assume you are a junior mathematician who has just found a proof of the Riemann hypothesis. You want to convince the mathematical world of your achievement, but you are extremely paranoid that if you reveal the proof to anyone (perhaps a senior expert), he or she will claim it was their own. While unlikely, this could be devastating to your career. Is there a way to prevent this from happening? Can you convince everyone that you know a proof without giving anyone a clue about it? Hold on.

The thrust of this section is not to prove more theorems, but rather to find proofs with additional properties. Randomized and interactive verification procedures (e.g., those in Section 10.1) allow the meaningful introduction of *zero-knowledge proofs*, which are proofs that yield *nothing* beyond their own validity.

Such proofs seem impossible—how can you convince anyone of anything they do not already know, without giving them any information? In mathematics, when we cannot prove a theorem ourselves, we feel that seeing a proof will *necessarily* teach us something we did not know, beyond the fact that it is true.

Well, the interactive proof given above, that two graphs are nonisomorphic, at least suggests that in some special cases, zero-knowledge proofs are possible! Note that in each round of that proof, the student knew perfectly well what the answer to his challenge was, so he learned nothing from the teacher's answer. In other words, *if* the graphs were indeed nonisomorphic (namely, the claim to be proved was true), the student could have generated the conversation with the teacher, *without* actually interacting with him! After all, in that case, the student knows the unique correct answer to every question he generated. And no new knowledge can be gained from a conversation that you can have with yourself. Despite this, the actual conversation taking place between them, a teacher's repeated success in identifying the correct graphs in many challenges, actually convinced the student that indeed, the graphs were nonisomorphic. In short, this interactive proof (at least intuitively) is a zero-knowledge proof.[7] It is convincing, and reveals nothing to the student that he didn't already know, beyond the truth of the claim itself.

How can we define this notion of *zero-knowledge proofs* for general interactive proofs? The motivation, formal definition, and some examples of this remarkable notion were given in the same seminal paper [GMR89] that defined interactive proofs. The definition is quite subtle and technical, and we only sketch the essence in high-level terms (more details are provided in Chapter 18 on cryptography). Extending the intuition from the example above, we can demand that on every correct claim, the verifier should be able to efficiently generate, *by herself*, (the probability distribution of) her conversation with the prover. This turns out to be unnecessarily stringent. Indeed, we would be satisfied if what the verifier can generate by herself is *computationally indistinguishable* from the actual conversation (defined formally in Section 7.3). In this sense, zero-knowledge proofs mean that no knowledge is leaked by the prover, which can ever be made use of by any efficient algorithm (like the verifier).

Now, which theorems have zero-knowledge proofs? Well, if the verifier can determine the answer with no aid, it is trivial. Thus, any set in $\mathcal{BPP}$ has a zero-knowledge proof, in which the prover says nothing (and the verifier decides by itself). A few examples believed to be outside $\mathcal{BPP}$, like graph nonisomorphism, are known to have such proofs unconditionally.

What is perhaps astonishing is that using the standard assumption of cryptography, namely, that *one-way functions* exist (see Section 4.5), then zero-knowledge proofs can be given for *every* theorem of interest! Goldreich, Micali, and Wigderson [GMW91] proved the following.

Theorem 10.5 [GMW91]. *Assume the existence of one-way functions. Then every set in $\mathcal{NP}$ has a zero-knowledge interactive proof.*

The "cryptographic" assumption is essential; a converse to this theorem was (formulated and) proved in [OW93].

The proof of the zero-knowledge theorem above again exemplifies the power of reductions and completeness. It is proved in [GMW91] in two parts. The first part gives zero-knowledge proofs (only) for statements of the form *"a given graph is 3-colorable"*.[8] The second part infers from the first that all $\mathcal{NP}$ sets have a zero-knowledge proof, by using the $\mathcal{NP}$-completeness of the graph 3-coloring problem.[9] We discuss both briefly.

---

[7]There is a subtlety, explained in [GMW91], which necessitates altering the original proof so as to formally make it zero-knowledge. This is an important subtlety, which arises from the need to prove zero-knowledge for "'cheating verifiers"; a verifier might deviate from the protocol (e.g. choose questions using a different probability distribution than is prescribed) in order to extract knowledge from the prover's answers.

[8]It is important to note that this first part uses *specific* combinatorial properties of the 3-coloring problem, in much the same sense as the combinatorial properties of graph nonisomorphism were used in its zero-knowledge proof.

[9]Specifically, it uses the strong form of reductions, mentioned after Theorem 3.13, which allows efficient translation of witnesses, not just instances, of problems in $\mathcal{NP}$.

For the first part, we need a protocol, in which a prover convinces a verifier that a given graph is 3-colorable, without revealing anything about the coloring (or anything else). The same protocol should guarantee that if the graph is not 3-colorable, the verifier will "catch" any cheating prover with non-negligible probability. The following simplistic description of the protocol captures the essence, but loses many important subtleties which are essential for formally guaranteeing these two important (and intuitively conflicting) properties. The main idea is this. The cryptographic assumption allows the prover to *commit* to a vertex coloring which perfectly *hides* it, akin to putting the color of each vertex in a separate sealed envelope. The verifier then picks a random edge in the graph, and asks the prover to "open" (as cryptography enables) the pair of envelopes on its endpoints, revealing the colors of these two adjacent vertices. If the graph was 3-colorable, a prover can generate a "sufficiently random" legal 3-coloring, which ensures that any pair of such revealed colors is completely random and distinct; this of course leaks no information to the verifier. If the graph was not 3-colorable, then at least one such pair of envelopes will contain clear evidence of the prover cheating (two equal colors, or possibly illegal contents), and cause the verifier to reject.[10]

For the second part, we need to give a reduction from proving arbitrary $\mathcal{NP}$ statements in zero-knowledge, using protocols doing so for 3-coloring graphs. Let us look at such a reduction in action, justifying the "application" above of the zero-knowledge theorem to (safely) convincing colleagues that you have made a mathematical breakthrough. Suppose that, indeed, you had proved the Riemann hypothesis, and were nervous to reveal the proof lest the listener rush to publish it first. With the zero-knowledge proofs of 3-colored maps, you could convince anyone, beyond any reasonable doubt, that you indeed have such a proof of the Riemann hypothesis, in a way which will reveal no information about it. You proceed as follows. First, use the efficient algorithm implicit in the proof that $3COL$ is $\mathcal{NP}$-complete to translate the statement of the Riemann hypothesis into a graph, and to translate your proof of it into the appropriate legal 3-coloring of that graph. Now use the protocol of [GMW91] for $3COL$ to convince your listener of this fact instead. Note that the listener could carry out the first part of the reduction (from Riemann hypothesis to a graph) by himself, and so knows that you are proving an equivalent statement.

But the grand impact of Theorem 10.5 is not in the (important) application to copyrights and plagiarism protection. Zero-knowledge proofs are a major tool for forcing participants in cryptographic protocols to behave correctly, that is, without compromising anyone's privacy [GMW91]. In essence (and glossing over numerous complications), zero-knowledge proofs allow parties in cryptographic protocols to convince others that their messages (computed partly depending on their private secrets) were computed in accordance with the protocol without revealing these secrets, by proving so in zero-knowledge (where these secrets are the proof that is never revealed). Let us elaborate the conceptual contribution of this and subsequent work to simplifying the task of protocol design for general cryptographic problems. Note that even defining the many intuitive notions below is highly nontrivial; the reader is again urged to look at the cryptography Chapter 18 for more detail.

The paper [GMW91] gives a (new kind of) reduction: It converts a protocol in which privacy is guaranteed only if all parties follow it,[11] and automatically generates a protocol in which privacy is guaranteed even if some parties are faulty or even maliciously deviate from protocol.[12] Soon afterward, Yao [Yao86] designed his celebrated *secure evaluation* protocol for honest parties (extended from 2 parties to any number of parties in [GMW87]). These protocols[13] offer yet a completely different reduction, converting an arbitrary circuit whose inputs are distributed among different parties into a protocol (for honest players) that evaluates this circuit on these inputs, without leaking any information to any subset of the players beyond what the output itself reveals. As a simple example from Yao's paper that might demonstrate this achievement, try designing such a protocol for two parties, each holding an n-bit integer, representing two millionaires trying to figure out who is richer without revealing their own worth. Another example is holding an election: n people each hold a bit (say), and try to figure out the majority vote. Again, everyone is honest and will follow every instruction of the protocol to the letter! They are simply curious, so the protocol should be designed so that

---

[10]Amplifying this non-negligible probability of catching the verifier is another important issue we do not discuss.

[11]Designing such a protocol for *honest* parties is a highly nontrivial task by itself, which we presently discuss.

[12]As in all such reductions, what is actually shown is that the ability of some parties to gain access to secrets of others entails an efficient algorithm to invert a one-way function.

[13]Which rely on the existence of trap-door functions.

no subset of them learns *anything* that the output itself does not reveal about the inputs of other people.

The combination of a secure evaluation protocol for any function that is private assuming honest players, and the compiler of such a protocol making it resilient against malicious parties, yields a *private and fault-tolerant* implementation of just about any cryptographic task. For a good example of the complexity of such tasks, which now become implementable, consider how a group of untrusting parties can play a game of poker *over the telephone*. No physical implements (like cards with opaque backs for poker) are allowed (or needed)—only digital communication and trap-door functions. As mentioned, this example and the general problem of secure function evaluation are discussed at length in the cryptography Chapter 18.

Since the original definition of zero-knowledge proofs, many variants suiting a variety of settings and applications proliferated, and ingenious protocols for them were developed. We mention here only one, perhaps the most surprising. Recall that the main result of this section, Theorem 10.5, establishes zero-knowledge *interactive* proofs of every theorem. As it turns out, one can define a model in which zero-knowledge proofs can be *non-interactive*, namely normal, written proofs! This model, along with the companion result that in it every theorem has such non-interactive zero-knowledge proof, appears in [BDSMP91].

10.3   Probabilistically checkable proofs (PCPs), and hardness of approximation

In this section, we turn to one of the deepest and most surprising discoveries about the power of probabilistic checkable proofs (PCPs) and their consequences for the limits of approximation algorithms.

We return to the non-interactive model, in which the verifier receives a (alleged) written proof, as in $\mathcal{NP}$. However, we now restrict the verifier's access to the proof, so as to read only a tiny part of it (which may be randomly selected). Indeed, it will turn out that only a constant number of bits of a written proof, appropriately formatted, are sufficient to verify with high probability that it is correct! This extremely counterintuitive PCP theorem, one of the major highlights of complexity theory, is the focus of this section.

It is remarkable to note that as natural as written, *non-interactive* proofs are to us, the model we discuss in this section arose very indirectly. It was derived from the multi-prover *interactive* proof model $\mathcal{MIP}$ of [BOGKW89] mentioned in Section 10.2. Recall that in $\mathcal{MIP}$ the verifier communicates with *several*, mutually non-communicating provers.[14]

The connection between multi-prover systems and written proofs is the simple and powerful observation of [FRS88]. They showed that $\mathcal{MIP}$, (*unlike* its single-prover sibling $\mathcal{IP}$), is *equivalent* to a non-interactive, written proof system (like $\mathcal{NP}$), but which (unlike $\mathcal{NP}$) restricts the verifier to a few random probes into the proof.[15]

An excellent, familiar setting of such "lazy verification" is when a referee is trying to decide the correctness of a long (say, 100-page) proof by sampling a few (say, 10) lines of the proof and checking only them. This seems useless; how can one hope to detect a single bug unless the entire proof is read? However, this intuition turns out to be valid only for the "natural" way of writing down proofs, in which single isolated bugs may indeed exist. Surprisingly, this intuition fails when *robust* formats of proofs are used (and, as usual, when we tolerate a tiny probability of error).

Such robust proof systems are called PCPs (short for "probabilistically checkable proofs"). Loosely speaking, a PCP system for a set $S \subseteq \mathbf{I}$ consists of a probabilistic polynomial-time verifier having access

---

[14]Curiously, the motivation of [BOGKW89] for defining $\mathcal{MIP}$ was also indirect, arising from cryptography and zero-knowledge proofs (discussed in the previous subsection). As mentioned there, for the single-prover proof system $\mathcal{IP}$, achieving non-trivial zero-knowledge necessitates having one-way functions. $\mathcal{MIP}$ was born in [BOGKW89] to provide a natural alternative proof system allowing *unconditional* zero-knowledge proofs. But now that it was born, complexity theorists had a new toy to play with (well, a new complexity class $\mathcal{MIP}$ to understand), and doing so created the snowball described below that eventually led to PCPs and their applications.

[15]The proof of this observation goes roughly as follows. In the easy direction, a multi-prover proof can be converted into a written one by writing down all provers' answers to all possible queries by the verifier. In the subtle direction (which does not work with a single prover), a written proof becomes the strategy of the (say) two provers, using which they respond to the verifier's queries. Of course, the verifier should not trust them, and so must make sure that they give consistent answers on the queries he would have made to the written proof. The fact that the provers cannot communicate underlies the soundness of this argument. Note that in both directions, the number of bit queries to the written proof is roughly the same as the number of communicated bits between the verifier and the provers.

to individual bits in a string representing the (alleged) proof.[16] On input x and an (alleged proof) y the verifier tosses coins and accordingly accesses only a *constant*(!) number of the bits in the alleged proof y. As in any proof system, it satisfies the usual completeness and soundness conditions, despite hardly looking at the alleged proof! For *every* $x \in S$, there is a sequence y (a correct proof) which the verifier accepts with probability 1. However, for *every* $x \notin S$, the verifier rejects with probability at least $1/2$, (no matter to which "alleged proof" y it is given).

A long sequence of ideas and papers, surveyed by Arora [Aro94] and Sudan [Sud96], in which the number of random probes to the written alleged proof was finally reduced to a fixed constant, culminated in the PCP theorem, a powerful new characterization of $\mathcal{NP}$, by Arora, Lund, Motwani, Sudan and Szegedy [ALM+98].

Theorem 10.6 (The PCP theorem [ALM+98]). *Every set in $\mathcal{NP}$ has a PCP system. Furthermore, there exists a polynomial-time procedure for converting any $\mathcal{NP}$-witness to the corresponding robust PCP-proof.*

Indeed, the proof of the PCP theorem suggests a new way of writing robust proofs, in which any bug must "spread" all over. Equivalently, if the probability of finding a bug found in these handful of bits scanned by the verifier is small (say, $\leq 1/10$), then the theorem is correct. The remarkable PCP theorem was proved with a rather complex and technical (mostly algebraic) proof, which resisted significant simplification for more than a decade. However, a conceptually different (combinatorial) proof that is very elegant and much simpler was given later by Dinur [Din07], using expander graphs (see Section 8.7) in an essential way.

10.3.1 Hardness of approximation

Since the 1970s, we've had an elaborate theory explaining the difficulty of optimization problems: for most natural ones, we knew if it was in $\mathcal{P}$ or $\mathcal{NP}$-hard. But the inability to efficiently find the exact optimum by efficient algorithms naturally leads one to seek an efficient approximation of the optimum. Numerous efficient approximation algorithms guaranteeing various approximation factors for different problems existed, but before the PCP theorem, we had no parallel theory to explain how good these were, or what the limits of efficient approximability are. This was a major conceptual and practical issue, with nearly no progress for two decades. And surprisingly, it was the advance on interactive proofs which was the source of progress on this major challenge.

The PCP theorem has revolutionized our ability to argue that certain optimization problems are not only $\mathcal{NP}$-hard to solve exactly, but they are $\mathcal{NP}$-hard even to get a rough approximation. Indeed, it led into a beautiful theory allowing, for numerous natural problems, to determine *exactly* how well they can be efficiently approximated.

The connection between the probabilistically checkable proofs and *hardness of approximation* was discovered by Feige et al. [FGL+96] and is elaborated on in the surveys mentioned above. We note that while we present this connection *assuming* the PCP theorem, it was actually discovered earlier, from (scaled down) versions of Theorem 10.4. This connection in turn presented a major motivation to the discovery of the PCP theorem!

Let us explore how one obtains an $\mathcal{NP}$-hard approximation problem from the PCP theorem. Consider the behavior of the PCP verifier on a given instance (say, of SAT). It can be described by a set of local tests on the given PCP proof; each test specifies the subset of bits to be read, and the set of values in these locations that would cause the verifier to accept. Now, if we consider the bits in a purported PCP-proof as Boolean variables, the question of acceptance by the verifier becomes a constraint satisfaction problem (CSP; see Section 4.3). The PCP theorem guarantees that either all constraints are satisfiable (if it was a "yes" instance) or that at most, $1/10$ of them are (if it was a "no" instance). This constitutes a reduction from SAT to this CSP. Approximating the maximum fraction of satisfied constraints in this CSP to within a factor < 10 would lead to an algorithm for solving SAT (exactly), and so approximating this CSP is $\mathcal{NP}$-hard.

This approximation problem seems contrived and perhaps does not arise in practice. But again, once we have it, we can try to use it and prove hardness of other, more natural approximation problems. As it happens, reductions between approximation problems are typically much harder to prove than standard

---

[16]In the case of $\mathcal{NP}$-proofs, the length of the proof is polynomial in the length of the input.

$\mathcal{NP}$-completeness results, and they often require significant analytic machinery. We mention two examples of the strongest such *inapproximability* results, both due to Håstad [Hås99,Hås01]. Both are nearly tight, in that it is $\mathcal{NP}$-hard to approximate the solution by the factor given, but it is trivial to do so with a slightly bigger factor. In both, $\varepsilon > 0$ can be an arbitrarily small constant.

- **Linear equations.** Given a linear system of equations over $\mathbb{F}_2$, approximate the maximum number of mutually satisfiable ones to within a factor of $2 - \varepsilon$ (clearly, a factor of 2 is trivial: A random assignment will do).

- **Clique.** Given a graph with n vertices, approximate its maximum clique size to within a factor $n^{1-\varepsilon}$ (clearly, a factor n is trivial: One vertex will do).

Despite these amazing results, pinpointing the exact limits of efficient approximation of many problems, for many other problems (e.g. *max-cut, vertex cover*), the best known approximation ratio does not match the (currently) best $\mathcal{NP}$-hardness result supplied by the PCP theorem. These gaps led to the development of the unique games problem and UGC, discussed in Section 4.3, which closes these gaps in numerous approximation problems.

10.4　Perspective and impact

It is impossible to overestimate the impact of the central idea of this chapter, namely to enrich the basic notion of *proof* with randomness and interaction. While the notion of a formal proof remains unchanged for establishing mathematical truth (including the truths proved in this chapter), interactive and probabilistic proofs (which do leave some chance of error) repay in numerous ways: they have properties which formal proofs cannot have, are extremely useful in practical situations, and give rise to a host of new theoretical developments. Here we briefly recount some of the main examples of this phenomenon, highlighting again how central these original proposals, coming out of cryptographic and group theoretic motivations, turned out to be elsewhere. Of course, one major impact area, *hardness of approximation*, was discussed in the previous subsection, and we list others.

Verifying computations, and the power of the prover in interactive proofs　Proof systems, essentially in all settings and applications where they come up, insist that the verifier is efficient; no matter how superior or ingenious the prover of a theorem was to come up with a proof, a mere mortal can check that it is correct. But various theoretical and practical considerations, which arose already as the theory of probabilistic and interactive proofs was developed, also questioned the power of the prover. Indeed, the prover may have to work much harder than the (complexity theoretic) difficulty of the statement. For example, to prove co$\mathcal{NP}$ statements is possible by Theorem 10.3, but the prover in that theorem is in $\mathcal{PSPACE}$ (and we suspect it cannot be weaker than in $\#\mathcal{P}$, a presumably much higher class than co$\mathcal{NP}$).

This question becomes ultra-important when a strong user (the prover) is trying to convince a weak one (the verifier) that a computation it performed is correct. This arises regularly in *delegation*, where the weak one (e.g. a smartphone) is letting a strong one (e.g. a cloud) perform a complex computation for it, but does not trust it. One hopes that convincing the weak one of correctness is not much harder than the computation itself. Delegation, which has seen much progress, is discussed in Section 18.9.2. A very different setting in which this need arises came with the advent of quantum computation: how can a classical verifier be convinced of the correctness of a computation performed by a purported quantum device? Again, convincing should hopefully not be much harder for the quantum algorithm than the computation itself. Quantum verification, which again had significant recent progress, is discussed in Section 11.3.

Locality in error-correcting codes　As PCPs are queried in just a few coordinates, and this somehow suffices for verifying their correctness, they are extremely robust to noise: changing a valid PCP in a sufficiently small fraction of its coordinates will almost surely not affect the answers to such random queries, and hence will be accepted by the verifier. In this robustness to noise, PCPs resemble error-correcting codes:

in a sense, a valid PCP can be derived from a noisy one just as a valid codeword can be derived from a noisy one, if the noise is sufficiently small. Indeed, coding-theoretic techniques were used in designing PCPs.

Then the impact went the other way, big time! The main feature of PCPs, namely their local checkability, was imported to the theory of error correction. Classically in coding theory, an entire message, corrupted by noise, is received and processed. With this novel, *local* view in mind, new error-correcting codes were sought and designed which allow a variety of tests and correct information extraction to be performed locally. This has led to definitions and constructions (see e.g. the given references and others they point to) and locally-testable codes [Gol11, GG16], locally-decodable codes [Yek12], locally-correctable codes [DSW14a], locally-recoverable codes [Maz18], private information retrieval schemes (PIRs) [CKGS98, DG16] and many others. Needless to say, these too have found practical applications as well as theoretical connections to the classical theory of error-correcting codes, computational complexity and other mathematical areas, e.g. combinatorial geometry [KS09, DSW14b]. Many beautiful open questions in this broad area remain, with perhaps the most stunning ones being the existence of constant query locally-testable codes of linear block-length and locally-decodable codes of polynomial block-length, which can support constant noise rate.

Property testing and sublinear-time algorithms   The idea of a few random spot-checks to test proximity of a sequence to the global property of being a codeword was greatly generalized in [GGR98], who focused on graph properties. This in turn was further extended to testing (proximity to) properties of sets, of probability distributions (connected to statistics), of hypergraphs, of functions, etc. An extensive text on the subject is the book [Gol17].

And while the above research focused on what can be done with a constant number of queries, techniques were developed to allow more generally probabilistic algorithms to sample much less than all the input data available to them, and use the resulting information to learn many globally-defined properties. This played extremely well with our era of data explosion (e.g. in biology, physics, Internet, ...), so vast that indeed there is no time to go over it all. This growing field is called *sublinear-time algorithms*, see e.g. the surveys [CS07, RS11].

Black-box vs. white-box access   The question of whether we can efficiently glean information or utilize a *given* computational device (say a software code or a piece of hardware) any better than simply having access to the function it computes (namely, to its input-output behavior) is fundamental to understanding computation. The second type is usually called *black-box* access, as we cannot peek into the device's inner workings. The first type is usually called *white-box* (or *clear-box*) access, where the complete description of the computational device is given. The effect of this basic issue is discussed in several chapters of this book, e.g., on pseudo-randomness in Section 7.3.3, and on program obfuscation in Section 18.9.3. It is fundamental to all efficient reductions (see Section 3.6) which essentially use one computation to create another. The relative power of black-box vs. white-box access has been studied in many other contexts.

A surprising power of white-box over black-box access in the context of cryptography was discovered by Barak, and essentially depends on the PCP theorem. Barak's PhD thesis [Bar04] gives several ingenious and powerful demonstrations where white-box access to a computation, *given in robust PCP form*, can drastically change our efficiency, or even our ability to achieve certain cryptographic primitives and tasks. This thesis has unleashed the area of "non-black-box crypto", which is beyond our scope. We just speculate that this idea may well find applications to computational complexity proper, possibly even lower bound proofs.

Practical probabilistic proofs and their applications   While many types of probabilistic proofs arose from potential applications, especially in cryptography and then in delegation, the original constructions as well as subsequent ones were too complex to even conceive of using them in actual systems. Remarkably, in the past decade, this has started to change! Some of the motivation came from new potential applications without trusted parties, and where anonymity can be essential. These include digital currencies (e.g. Bitcoin, Zcash, and Etherium) and public ledgers, and more importantly the underlying (buzzword) *blockchain* which serves as their infrastructure. These new protocols required significant theoretical work, and we give only a couple of references the reader can use to start exploring: see e.g. [BSCTV17] on zero-knowledge proofs,

and [BSBC+17] on PCPs. Needless to say, when protocols indeed become efficient enough, many of the general applications discussed in the cryptography Chapter 18 may become reality as well!

But probabilistic proofs developed in other unexpected practical directions. We restrict this paragraph to zero-knowledge proofs, where the variety is stunning. Indeed, perhaps this should not be too surprising; the counterintuitive ability to be convincing without revealing information has such a universal appeal that the readers should easily find examples from their daily lives where such ability would be not only useful, but even essential. And it is! But unlike the protocols we discussed in this chapter, which are digital (namely, exercised by computers exchanging bits), in the real world the agents may be people, attempting to argue about properties of physical objects, and using physical objects and the laws of nature in their arguments. One remarkable example was the protocol for verification of nuclear warhead reduction between untrusted parties [GBG14]. Another is for private comparison of DNA sequences (say for criminal investigations or genetic testing); this and many other examples, as well as a higher level view of "physical" zero-knowledge proofs are given in [FFN14].

11 Quantum computing

This chapter describes a unique and exciting interaction between computational complexity and physics that probes the nature of reality and brings a new perspective to its study. We will discuss some of the many facets of this interaction. There are many excellent texts on this broad subject, including [KSV02, NC10, Aar13a], each with a different perspective and style.

Let us return to the most basic question: What are the problems that can be solved efficiently? In the beginning of this book, we defined them to be in the class $\mathcal{P}$, those problems solvable in *deterministic* polynomial time. Rapid progress in computer technology made them run faster, but the class $\mathcal{P}$ remained robust under all models. The first potential change occurred when people realized that we use *randomness* in algorithms. It seems that nature provides us with unlimited free random bits, and we can incorporate them into the computations of Turing machines. This allows us (if we are willing to tolerate errors with small probability) to broaden the class of tractable problems to those having polynomial time *probabilistic* algorithms, namely, the class $\mathcal{BPP}$. While we don't know if randomness really buys extra power (and conjecture that it doesn't—see Section 7.2), many probabilistic algorithms are in current use, simply because the best deterministic algorithms we currently have are much (often exponentially) slower.

It stands to reason that we should add to our computers and algorithms everything nature provides that seems to increase their power. Indeed, if our computers cannot efficiently simulate some natural phenomena, we should integrate in them the underlying mechanism enabling nature to be more efficient. Feynman [Fey82] has noted that the obvious classical algorithm (even using randomness) for simulating the evolution of a quantum system on n particles requires exponential time in n. He thus suggested that computer algorithms should be equipped with "quantum mechanical" gates, to enable such efficient simulation (and possibly make them more powerful than classical computers). A similar idea was put forth in Russia by Manin [Man80]. A series of papers [Ben80, Deu85, Yao93, BV97, AKN98] then completely formalized the concept of a quantum mechanical Turing machine, which I informally describe below. Restricting it to run in polynomial time, we get the class $\mathcal{BQP}$, of functions efficiently computable by such algorithms. Observe that like $\mathcal{BPP}$, $\mathcal{BQP}$ also allows small errors (which again can be made arbitrarily small by repetition).

Definition 11.1 (The class $\mathcal{BQP}$ [BV97]). A function $f: \mathbf{I} \to \mathbf{I}$ is in $\mathcal{BQP}$ if there exists a quantum polynomial-time algorithm A, such that for every input x, $\Pr[A(x) \neq f(x)] \leq 1/3$.

We shall return to discuss the power of this class. First let me explain what a quantum algorithm is.

How does a quantum algorithm work? It turns out that to understand that, one does not need to know any quantum mechanics.[1] Let us explain them in analogy with deterministic and probabilistic algorithms that we have already met. It would be useful to do so from the viewpoint of how the entire "state" of each of these types of algorithm evolves over time. Every algorithm evolves a state—the content of its memory of (say) n bits—via a sequence of local operations (each acting on a few bits). The different types of algorithms differ in the nature of a state, the local operations allowed, and the definition of the output of that process. In all of them, the initial state contains the input to the problem in a designated location in memory (with all other bits set to a default value, say, 0).

In a deterministic algorithm, the state is simply the value of these bits, a vector $x \in \{0,1\}^n$. A single operation picks, for example, three of them and uses their current value to replace them with a new value, according to any function $g: I_3 \to I_3$. For example, one such 3-bit function, called the *Toffoli gate*, writes in one bit position the XOR of its current value with the AND of the other two bits (this single function can simulate the standard Boolean gates $\vee, \wedge, \neg$). When the machine halts, the output of this computation is the contents of some designated subset of the bits. Summarizing, the evolution of a deterministic algorithm takes place in the discrete space $x \in \{0,1\}^n$.

In a probabilistic algorithm, the local operation can be probabilistic. For example, in one step it can apply, with probability, say, $\frac{1}{4}$, the Toffoli gate to three specific bit locations, and with probability $\frac{3}{4}$ leave the bits intact. Accordingly, the state of the algorithm over time is a random variable, which may be viewed as a *convex combination*[2] $\sum_{x \in \{0,1\}^n} p_x x$ over n-bit sequences x. So the local operations evolve the probability

---

[1] Indeed, quantum algorithms may be viewed as a language to explain many of the principles of quantum mechanics.

[2] Namely, $p_x \geq 0$ for all x, and $\sum_{x \in \{0,1\}^n} p_x = 1$. In other words, p is a nonnegative vector of unit norm in L_1.

vector $p = (p_x)$ over time. The output again resides in some specified bit locations but is now a random variable over Boolean vectors as well (distributed as the marginal of the distribution in these locations). So the evolution of a probabilistic algorithm takes place in $\mathbb{R}^{2^n}$, with the vectors $x \in \{0,1\}^n$ serving as a natural basis for this space.

In a quantum algorithm, the state is again viewed as a linear combination (called a *superposition* or a *wave function*) $\sum_{x \in \{0,1\}^n} \alpha_x x$, only now the coefficients can take complex values, and the vector of coefficients (called *amplitudes*) $\alpha = (\alpha_x)$ must have unit norm in L_2.[3] A local operation can take, as before, some constant number of bits (called *qubits* in this setting) and perform a (norm-preserving) *unitary* linear operation on them (which, to formally be an action on the entire state, is tensored with the identity operator on the remaining qubits). An important example is the Hadamard gate, acting on one qubit. Being a linear operation, we can describe it by its action on a basis: It sends the state $\mathbf{0}$ to $(\frac{1}{\sqrt{2}}\mathbf{0} + \frac{1}{\sqrt{2}}\mathbf{1})$ and sends the state $\mathbf{1}$ to $(\frac{1}{\sqrt{2}}\mathbf{0} - \frac{1}{\sqrt{2}}\mathbf{1})$. It remains to define the output of an algorithm, which like the input, should again be a Boolean vector. This is obtained by *measurement*, which we now define. Assume for simplicity the output is the full contents of the memory, and the final state is $\sum_{x \in \{0,1\}^n} \alpha_x x$ in $\mathbb{C}^{2^n}$. As α is a unit vector, the vector $p_x = |\alpha_x|^2$ is a probability vector, and the output is defined to be x with probability p_x.[4] Summarizing, the evolution of a quantum algorithm takes place (on the unit sphere) in $\mathbb{C}^{2^n}$, after which a measurement converts the final state into a probabilistic output.

A few comments are in order. First, it seems that the number of possible gates is infinite, but in fact, just as for classical computation, a finite set of elementary operations suffices. Indeed the (classical) Toffoli gate and the (quantum) Hadamard gate together form a universal set of gates. Second,[5] it turns out that complex numbers are not really essential; real numbers suffice, as long as they can be also *negative*—as we'll see, this seems to be the source of power of quantum algorithms over probabilistic ones. Finally, quantum algorithms can toss coins: note that applying the Hadamard gate to a single fixed qubit (say, 0), and then measuring the outcome results in a perfect coin toss, which is 0 with probability $\frac{1}{2}$ and 1 with probability $\frac{1}{2}$. It follows that quantum computing can simulate probabilistic computing.[6] This last point leads to the following theorem.

Theorem 11.2. $\mathcal{BPP} \subseteq \mathcal{BQP}$.

So, what else is in $\mathcal{BQP}$? Shor's breakthrough paper [Sho94] supplied the most stunning examples, integer factoring and discrete logarithms, discussed in Section 4.5.

Theorem 11.3 [Sho94]. *Factoring integers and computing discrete logarithms are in $\mathcal{BQP}$.*

How do these algorithms work? What do they do that classical algorithms can't? Both quantum and probabilistic algorithms seem to "simultaneously" act on all 2^n binary sequences, so this alone is not the source of power. The buzzword answer is *interference*, which should be understood in the same intuitive sense you learned in high school, of (ocean, electromagnetic, sound, etc.) waves interfering (constructively or destructively). Having negative coefficients means that quantum algorithms can generate cancellations at this exponential scale, which decreases or eliminates the probability of unwanted outcomes, and thus increases the probability of desired outcomes by unitarity. This interference is something that cannot happen in probabilistic algorithms, as all coefficients of the state vector are nonnegative for these algorithms. Thus, probabilistic algorithms in practice evolve just a *sample* of the probability distribution (as opposed to physically maintaining the exponentially long state vector), whereas for a quantum algorithm to work, the whole superposition has to "exist" and evolve.

Let us try to probe this magical power of interference a bit, discussing a key aspect of Shor's algorithm. An important subroutine of that algorithm computes the *discrete Fourier transform* (DFT) of the state, over an exponentially large Abelian group $\mathbb{Z}_N$. How can one compute an exponentially large linear transformation

---

[3]Namely, $\sum_{x \in \{0,1\}^n} |\alpha_x|^2 = 1$.

[4]If the output is designated to only be a subset of the bits, we similarly give each Boolean output a probability that is the total square length in α of full sequences containing it.

[5]Which follows from the first point but can also be seen directly.

[6]While we only allowed a measurement at the very last step of quantum algorithms, one can define them alternatively to allow measurements at any step—this does not change their power—and so they can toss coins at any step.

in polynomial time? Let me demonstrate it by a simpler subroutine used in Simon's algorithm [Sim97] for computing the DFT over the Boolean cube $(\mathbb{Z}_2)^n$ (indeed, this work inspired Shor). The algorithm takes only one line: Simply apply the Hadamard gate on each of the n bits in sequence. Please check that the tensor product of the n 2×2 matrices describing the Hadamard gate is the $2^n \times 2^n$ matrix describing DFT over $(\mathbb{Z}_2)^n$. To see interference in action, consider the output of the DFT when applied to a state vector in which all states have the same amplitude (which is $2^{-n/2}$). It is not hard to see that the result is a state in which the all-0 vector has a nonzero (indeed, 1) amplitude, and amplitudes of all other vectors become zero. So, the amplitude of this all-0 vector increased exponentially by constructive interference, whereas the amplitude of all other vectors was diminished to zero by destructive interference (of course, this is just an intuitive description of a simple fact in linear algebra). In a similar but far more sophisticated manner, Shor applies the DFT over $\mathbb{Z}_N$ (where N depends on the integer to be factored) to evolve a certain state vector, and interference causes the output to be (concentrated near) the nontrivial factors of that integer (if they exist) with large amplitudes, while diminishing all other (exponentially many) alternative states.

What other problems can be solved efficiently by quantum algorithms for which classical ones are not known? The general technique of Grover [Gro96] offers a quadratic quantum speed-up over classical algorithms for abstract search problems. But a possible super-polynomial gap, as Shor's algorithm may provide, has only relatively few additional examples, most of which also have a similar number-theoretic or algebraic flavor (see, e.g., the survey [CvD10] and the more recent [EHKS14]). In many, the essence of the algorithms is finding some periodic structure using a *fast Fourier transform* (FFT) algorithm in some appropriate group, where "fast" means (classical) time $N(\log N)^c$, and N is the size of the group. The quantum parallelism described above shaves off the factor N (which is typically exponential in the input size n), resulting in a quantum algorithm of time complexity $(\log N)^c = \text{poly}(n)$. All Abelian groups have a FFT, and it is natural to attempt to generalize such FFTs for non-Abelian groups (which instead of the characters compute the irreducible representations) to possibly solve other problems. A key challenge of this type is the graph isomorphism problem (which has been mentioned in this book a few times, see e.g. Section 4.2). While the (seemingly required) quantum version of the FFT is known [Bea97] for the group in question (the symmetric group S_n), we have no idea how to use it for an efficient quantum algorithm for graph isomorphism.

Another challenging task is the invention of new ways to use quantum interference, possibly for problems of a different nature. One interesting technique, which has yet to suggest a possible quantum exponential speed-up over classical algorithms, is the *quantum walk* on graphs, which was initiated by Farhi and Gutmann [FG98] with a great many follow-up works (see [CCD+03] for a provable exponential gap in a black-box model). A completely different proposal of Aaronson and Arkhipov [AA11] is to utilize the behavior of non-interacting bosons in linear optics to efficiently solve a certain sampling problem related to the permanent function (which will play a major role in Chapter 12), which has no efficient classical algorithms (and is even outside of $\mathcal{NP}$) under a natural conjecture.

Summarizing, the power of quantum algorithms in comparison to classical ones is far from understood. On the one hand, practically no one believes that they can solve $\mathcal{NP}$-complete problems. In symbols:

Conjecture 11.4. $\mathcal{NP} \nsubseteq \mathcal{BQP}$.

On the other hand, almost everyone believes quantum algorithms are stronger than probabilistic algorithms, namely:

Conjecture 11.5. $\mathcal{BPP} \subsetneq \mathcal{BQP}$.

We know just a bit more about the finer placement of the complexity class $\mathcal{BQP}$ of efficient quantum algorithms in the chain of classical complexity classes between $\mathcal{P}$ and $\mathcal{PSPACE}$ of Figure 10, defined in Section 4.1. On the one hand, $\mathcal{BQP}$ can be efficiently simulated with access to counting queries $\#\mathcal{P}$ (essentially, since counting allows adding the exponentially many amplitudes in a quantum state). On the other hand, a recent breakthrough of Raz and Tal [RT18] gives an *oracle*[7] making $\mathcal{BQP}$ solve problems outside the polynomial hierarchy $\mathcal{PH}$.

We turn to discuss the practical feasibility of $\mathcal{BQP}$.

---

[7]See Section 5.1 and this paper for the meaning of oracle separation.

11.1   Building a quantum computer

Besides being a great theoretical achievement, Shor's algorithm had a huge practical impact on quantum computing. Recall that factoring and discrete logarithms underlie essentially all cryptographic and e-commerce systems today, and so everyone wanted a quantum computer.[8] However, the power can also give rise to new types of cryptographic schemes, secure against stronger attacks. This is the subject of quantum cryptography, which we will not discuss here. Therefore, after two decades of purely academic interest in quantum computing, Shor's paper suddenly incentivized governments and industry to invest billions of dollars in developing a working quantum computer, with remarkable new technologies already developed that have made progress in overcoming significant obstacles. One concrete (though biased) way to measure the quality of continually improving technologies and designs is by the largest integer they can factor using Shor's algorithm. Today, two decades after Shor's paper, the largest number to be factored so far this way is 21.

What problems are encountered when building a quantum computer? After Turing defined his Turing machine, the locality and simplicity of his design instantly suggested implementations. And despite the bulky and faulty technology at the time (this was the pre-transistor period—each bit needed its own vacuum tube!), large-scale working computers were quickly built. Today the incomprehensibly fast progress of technology results in speed and memory improvements that make us consider last year's smartphone an archeological find. But this progress is all classical. The state of a classical computer is always just a sequence of bits. The same is true for probabilistic algorithms, which simply evolve a *sample* of the distribution describing the current probabilistic state. However, if we want *interference*, there is no way to sample the state of a quantum computer (e.g., by measurement) without destroying the information. So, what is difficult is holding many bits in a complex *entangled* state.[9] This becomes even harder when this quantum state is evolving under computation. Another significant problem to overcome is *decoherence* noise—in quantum mechanics, everything is invariably entangled with everything else, and the state of a quantum computer is affected by the outside quantum world. The important quantum error-correcting codes invented by [Sho95, ABO97, Kit03] and others[10] to combat decoherence will be used. But it is not clear that they suffice in practice, or even that the model of noise they assume reflects physical decoherence. Diverse ingenious technologies are competing in many projects around the world to achieve this goal, and despite technological breakthroughs with other benefits, progress toward a large-scale universal quantum computer still seems slow.

Is there an altogether different problem preventing progress? What can it be? After all, the existence of a quantum computer is entirely consistent with the theory of quantum mechanics (and actually uses only rudimentary parts of that theory). Well, perhaps the theory is wrong, or at least incomplete! Perhaps quantum mechanics needs a revision to handle a very large number of entangled particles, in analogy to the revision needed in Newtonian mechanics to handle bodies moving at very fast velocities, or interacting at very high energies and tiny (e.g., Planck scale) distances? Such a revision may put limits on what quantum computers can do and may explain the above mentioned slow progress. This is a fascinating state of affairs, and it seems that regardless of the outcome, we'll understand more about both quantum physics and computer science as scientists of both fields join forces to grapple with these problems.

But to whet your appetite for what's coming soon, consider this. When some company tries to sell you a "quantum computer", how would you know it is? More precisely, how can a classical algorithm verify computation performed by a *computationally superior* quantum algorithm? This fundamental scientific issue is discussed below in Section 11.3.

And while general purpose quantum computers are not here yet, the technological progress achieved by these practical projects on quantum computational technologies yielded new ways to further Feynman's

---

[8]We note that while a quantum computer can destroy some classical hardness assumptions for cryptography, it is not known to destroy them all (e.g., the ones based on lattice problems, such as those mentioned at the end of Section 13.8).

[9]Entanglement is the quantum analog of probabilistic correlation. Informally, just as the correlation among the bits in a probability distribution on bit sequences captures how far this distribution is from a product of independent distributions on individual bits, the entanglement among the qubits in a quantum state captures how far this state is from a tensor product of quantum states on the individual qubits. It should be stressed that entanglement can be far more complex, and is far less understood, than classical correlation.

[10]Such results often go by the name "quantum threshold theorem" to indicate that noise below a certain constant threshold per bit can be tolerated in arbitrarily long quantum computations.

original motivation, of efficiently simulating *some* quantum systems. In addition, such progress has also yielded implementations of quantum protocols for *some* cryptographic problems whose properties follow from quantum mechanics and hence can resist computationally unbounded adversaries. This is in contrast to classical protocols for the same problems, which need to rely on computational assumptions and can withstand only computationally bounded adversaries.

11.2   Quantum proofs, quantum Hamiltonian complexity, and dynamics

We return to the mathematical arena and conclude this section with a few directions in which significant theoretical progress has been made. I will not elaborate on the many advances in quantum information theory and quantum cryptography, by now very developed disciplines, with actual applications to boot (for example, the theoretical ideas of quantum teleportation [BBC$^+$93] and of quantum key distribution [BB84, Eke91] are becoming a reality). Instead I wish to focus on quantum complexity theory, namely, the application of the methodology we have seen in action in the previous chapters of generalization, reduction, and completeness. This study often happened in beautiful interactions with physicists, and with direct consequences for physics problems as well as for the philosophy of science.

A central notion studied in the classical world is *proof*, and having a new model of efficient computation, it is natural to extend proof (and other notions) to this setting. What is a proof in the quantum world? The most basic notion is a natural generalization of $\mathcal{NP}$ to the quantum setting, in which the verifier is a $\mathcal{BQP}$ machine, and the witness is allowed to be a (short) quantum state. The class defined by all problems that have such a verification system is called $\mathcal{QMA}$.[11] In analogy to Cook's and Levin's discovery of a natural $\mathcal{NP}$-complete problem, SAT, Kitaev [Kit03, KSV02] discovered a $\mathcal{QMA}$-complete problem, which is natural from both the computer science and physics standpoints. It can be naturally viewed as a quantum *constraint satisfaction problem (CSP)*, analogous to 3-SAT, and moreover, the local constraints arise naturally from quantum Hamiltonian physics. We note that Kitaev's quantum reduction is quite a bit trickier than the Cook-Levin classical one, despite the syntactic similarity. The reason is that for classical states, local consistency (of the evolution of computation) implies global consistency, whereas in the quantum setting, two globally very different superpositions can look the same locally (e.g., when projected on every three qubits). Still, local consistency checks turn out to suffice, and we now go over some consequences and continuations of Kitaev's work (details can be found, e.g., in the survey [GHL14]).

We start by defining the notions of a quantum Hamiltonian and of a quantum *local* Hamiltonian. First, a *Hamiltonian* (on n qubits)[12] is simply a Hermitian $2^n \times 2^n$ matrix H. In quantum mechanics, Hamiltonians define the dynamics of an n-body quantum system state Ψ over time t via Schrödinger's equation: $i\hbar \frac{\partial \Psi}{\partial t} = H\Psi$. A Hamiltonian H is *local* if it is the sum $H = \sum_i H_i$, where each H_i is a Hermitian matrix acting on a constant number of qubits (in the same way the local gates act in quantum algorithms). Note that each such H_i is a "local constraint"—it is essentially a constant-size matrix[13] and so can be described concisely: naming the indices of the qubits it acts on, and how it acts on them. Many local Hamiltonians arise from quantizing statistical mechanics models of local interactions, like the Ising spin glass on a 2-dimensional lattice. A central problem in condensed matter physics is to determine its lowest eigenvalue (the *ground state energy*) and more generally, to understand its eigenvector (the *ground state* itself). We discuss in turn both of them, and how the computational perspective affected their study in physics.

11.2.1   The complexity of ground state energy

Kitaev's result above that quantum CSP is complete for $\mathcal{QMA}$ is precisely about computing (or rather, approximating) the ground state energy of a local Hamiltonian. Some of the important parameters of a wide class of quantum CSPs are the *geometry* (which subsets of variables interact), *locality* (the maximum size of such subsets), and *dimension* (the number of values each qudit can take in the H_i's). For example, Kitaev's is a 5-local CSP of dimension 2. For decades, physicists have studied numerous quantum CSPs arising in

---

[11]It would seem that $\mathcal{QNP}$ is a better name, but there is a reason for this notation, which I do not go into here.

[12]Or more generally *qudits*, which can take more than 2 values.

[13]Which is formally tensored with a huge identity matrix trivially acting on the remaining qubits.

a variety of natural settings. But the computational lens has had a significant impact on the way they have been studied. To understand why and how, observe first that any classical CSP (like 3-SAT) can be viewed as a quantum CSP. This suggests that the web of reductions and completeness results we have in the classical setting can be extended to the quantum world, allowing an understanding of the relative complexity of this minimum-energy problem for various quantum CSPs (and revealing, like optimization problems in the classical setting, many surprising connections). Such reductions, often between Hamiltonians from physically very different quantum systems, gave birth to the field of *quantum simulations* [CZ12], which allows studying one quantum system using another via such reductions. A large number of papers determined precisely the complexity of finding the ground state energy of many physically important Hamiltonians, and how it depends on the parameters of the Hamiltonian. This study has led to a rather complete characterization of the algorithmic difficulty of quantum CSPs in many cases. One very general result (see [CM13], completed by [BH14]) of this type determined the complexity of all CSPs of dimension 2 (namely, on qubits). As it turns out, these can be in $\mathcal{P}$, $\mathcal{NP}$-complete, $\mathcal{QMA}$-complete or "Ising-complete." This yields a quantum analog of the *dichotomy theorem*, Theorem 4.3, discussed in Chapter 4 for classical CSPs. Naturally, approximation algorithms and hardness were studied as well.

11.2.2   Ground states, entanglement, area law, and tensor networks

Now we turn to understanding and computing the ground state itself of a local Hamiltonian. First let us ponder: What does the problem even mean? After all, as discussed, for a system of n qubits, this is a vector in 2^n dimensions (with complex coefficients, no less). So, to be efficient, we can only hope to compute some succinct representation of the ground state (or an approximation of it), assuming one exists. Such succinct representations, which allow in particular the computation of local observables (e.g., the energy of the state) were suggested by physicists, with the primary one being *tensor networks*. Without describing them formally (see this friendly physics survey [Orú14], and this computationally centered one [AKK17]), they may be viewed as computational devices (like circuits) in that they "compute" the ground state much as circuits compute Boolean functions. In both cases, the obvious description of these objects has length exponential in n, but for some ground states, as for some functions, the computational description may be much more succinct (e.g., polynomial in n). As in computational complexity, understanding which local Hamiltonians possess such descriptions, and furthermore, finding these efficiently from the description of the Hamiltonians, are central questions.

An efficient tensor network necessarily restricts the entanglement in the state it represents; its geometry implies that certain subsets of the particles cannot be too entangled with other subsets. So an even more basic question is: Which Hamiltonians have ground states with such limited entanglement structure? It stands to reason that the geometry of the local interactions of the Hamiltonian affects the structure of these quantum correlations (and may inform the construction of the tensor network). A major conjecture, called the *area law* conjecture, asserts that for any *gapped* system,[14] the entanglement of the ground state between any two sets of particles in the system is proportional to the *area* (or, graph theoretically, the cut size) of the local interaction graph between the parts. Thus, in 1-dimensional systems (where qubits reside on a line and all interactions are between neighboring points on this path of particles), entanglement should be bounded by a constant. In 2-dimensional systems (e.g., where n qubits reside on the vertices of a plane lattice or more generally, on the vertices of any planar graph), it should scale like $\sqrt{n}$. If we have an arbitrary graph of interactions, the area is measured by the number of edges crossing the relevant cut between the two parts, which by the area law conjecture, bounds the entanglement between them. The general area law conjecture is wide open, but there are exciting developments and interactions that we briefly discuss now.

First, let us discuss the existence of small tensor networks and then move to finding them. An important result of Hastings [Has07] is proving the area law for 1-dimensional systems. Further work [AAVL11], using some computational methods developed for a quantum analog of the PCP theorem (see Section 10.3),

---

[14] "Gapped" means that there is a constant spectral gap between the ground and the second-lowest energy levels of the Hamiltonian $H = \sum_i H_i$, when each local term H_i in it is normalized to have at most unit norm. This is a different regime than the $\mathcal{QMA}$-complete problems, where the gap is typically inverse polynomially small.

provided an exponential improvement to Hastings' bound.[15] As it turns out, Hastings' result implies the existence of a small (polynomial-size) tensor network (which for 1-dimensional systems is called a *matrix product state*). This raises the question of finding such a description efficiently. Indeed, this important question was considered much earlier, and several heuristic algorithms were developed, of which the most popular is the density matrix renormalization group, developed by White [Whi92]. DMRG and its variants are an extremely useful heuristic for many naturally occurring and studied physical many-body systems, but no theoretical explanation or provable bounds on their performance exist.[16] A sequence of papers using a variety of computational techniques culminated in [LVV13], which developed a completely different, provably polynomial-time algorithm, which constructs a matrix product state approximating the ground state of every gapped 1-dimensional Hamiltonian.

11.2.3   Hamiltonian dynamics and adiabatic computation

Let us conclude this overview with a short discussion of quantum Hamiltonian *dynamics*, as in Schrödinger's equation. It turns out that this dynamics suggests a new way to do quantum computing, suggested by [FGGS00], called "adiabatic computation." It is based on the adiabatic theorem of Born and Fock [BF28], one of the early and basic results in quantum mechanics. I give a high-level description of adiabatic computation, specifically to contrast it with our description of how a quantum Turing machine computes. By Kitaev's completeness result, essentially any problem we want to solve can be encoded as finding the ground state energy of a given local Hamiltonian, say, on n bits (for simplicity, you can consider the Hamiltonian encoding an instance of SAT). Call this Hamiltonian $H(1)$. Next, "prepare" some simple local Hamiltonian on n bits, and initialize it to its ground state. Call it $H(0)$. Finally, let $H(t)$ describe an evolution of $H(0)$ to $H(1)$, which can be any continuous interpolation between the two. The adiabatic theorem ensures that if this deformation is slow enough, $H(t)$ will stay at its ground state throughout the process, and so we will end up with the ground state of $H(1)$! This would solve our original problem (e.g., it will give a satisfying assignment of the encoded SAT formula). The cleverness of such a design is in picking the initial $H(0)$ and the evolution path $H(t)$ so that the ground state energies $e(t)$ of all Hamiltonians $H(t)$ are *well separated* from the next-higher energy level, say, by some $\gamma > 0$. By the adiabatic theorem, it suffices that evolution speed is slow enough that t moves from 0 to 1 in time $1/\gamma^2$. Can SAT be solved in polynomial time by this model? Can integer factoring?

The first remarkable thing to observe is that quantum mechanics offers many ways to compute, which look very different from each other. The next remarkable finding is a theorem [AvDK$^+$08] showing that the two computational models are equivalent!

Theorem 11.6 [AvDK$^+$08]. *Adiabatic computation and quantum circuits can simulate each other efficiently.*

As in the early days of classical computing theory, where many different types of computational models (1-tape Turing machines, multi-tape Turing machines, lambda calculus, random access machines, cellular automata, etc.) were found to be equivalent, Theorem 11.6 gives confidence in having the right computational model in the quantum setting as well. Indeed, yet another very different (but polynomially equivalent) universal model is the *topological quantum computer* proposed by Freedman, Kitaev, Larsen, and Wang [FKLW03] where logical gates made of *quantum braids* act on 2-dimensional quasiparticles called *anyons*.[17] This topological model possesses strong fault-tolerant properties and serves as a basis for some of the practical projects for building a quantum computer.

---

[15]The conjecture is wide open for 2-dimensional lattices. A very nice, seemingly much simpler challenge is to extend this constant upper bound on entanglement from paths to trees, where all cuts are still of size 1.

[16]This may be likened to the success of the simplex method for linear programming on many practically arising systems of linear inequalities. And like that story, where eventually a completely different algorithm (the ellipsoid method) was found to solve linear programming on all instances in polynomial time, here too there was a (theoretical) happy ending.

[17]Fear not—I too don't understand the last sentence.

11.3 Quantum interactive proofs, and testing quantum mechanics

Just as one can generalize *written* $\mathcal{NP}$-proofs, one can attempt to generalize the *interactive* proofs we discussed in Section 10.1 and study their power. A variety of analogs of the interactive proof systems have been defined, yielding new characterizations of these classes in terms of classical complexity classes and analog theorems to classical ones (see, e.g., [BFK10, JJUW10, IV12, BJSW16]).[18] But one of the most intriguing developments regarding interactive proofs was found in studying the following "mixed" proof system. The prover in this system is not arbitrarily powerful (as, e.g., in $\mathcal{IP}$), but rather is an efficient quantum algorithm (namely, in $\mathcal{BQP}$). The verifier in this system is a classical algorithm (in $\mathcal{BPP}$). Let us discuss the fundamental motivation for such mixed proof systems.

Is quantum mechanics a falsifiable scientific theory? This basic question should be asked of any theory, and the general scientific paradigm for studying it is sometimes called "predict and experiment." Any theory predicts the outcome of certain experiments or observations. When the observations match prediction, we further validate the theory. If they don't match, the theory is wrong and needs revision. Quantum mechanics, being so paradoxical and counterintuitive, has generated a host of suggested experiments and actually withstood many. Einstein, who refused to believe it all his life, devised many such experiments, of which perhaps the most famous is the EPR "paradox" [EPR35] proposed by Einstein, Podolsky, and Rosen in 1935. They challenged the possibility, suggested by quantum mechanics, that quantum information is apparently maintained and communicated instantly, faster than the speed of light. In the 1960s, Bell's famous inequalities [Bel64] suggested a concrete experiment (simplified by [CHSH69]) to refute the "local hidden variable" interpretation of quantum mechanics. Such experiments were successfully conducted starting in the 1980s (see [Asp99]), in some sense certifying that quantum mechanics is indeed paradoxical and counterintuitive, or at least defies simple classical probabilistic explanations. And while quantum mechanics is completely accepted and is used to great technological impact, philosophical arguments about its interpretation persist today. As mentioned above, the possibility of large scale, general purpose quantum computing is a real challenge to how complete this theory is. How can one test this?

Well, if indeed quantum algorithms can be exponentially faster than classical algorithms for some problems, then every such algorithm suggests a new experiment to conduct, and with it, a real dilemma. Take any function that possesses a fast quantum algorithm but no classical one. The prediction to test is simply that the given quantum algorithm computes the correct answer (on some input). The dilemma is how to test this fact by efficient classical means, for which the correct answer is, by assumption, impossible to obtain. This is no mere philosophical problem! It may become a real issue if and when one of the multiple projects attempting to build quantum computers will claim success (as do already some companies who sell computers allegedly using quantum algorithms). How do we test such claims? Of course, for some problems, a purported quantum algorithm can be efficiently tested classically. For example, Shor's factoring algorithm is easily verifiable, as it produces factors that a classical verifier can multiply back and check. More generally, such classical testing can be done for any function in $\mathcal{NP} \cap \mathrm{co}\mathcal{NP}$, if the quantum algorithm produces the necessary witness. But $\mathcal{BQP}$ potentially contains much harder problems (e.g., outside $\mathcal{NP}$). How can we test quantum algorithms (or devices) purporting to solve these problems?

A new idea, put forth independently in [BFK09, ABOE10] (see full proofs and the historical survey in [ABOEM17]), is to allow *interactive* experiments, in the spirit of interactive proofs (discussed in Section 10.1). Namely, assume that a $\mathcal{BQP}$-algorithm can solve some problem. Can (possibly another) $\mathcal{BQP}$ algorithm interact with a classical $\mathcal{BPP}$ algorithm and convince it (with high probability) that it solves this problem correctly? Can such interactive verification be done for *all* problems in $\mathcal{BQP}$? Aharonov, Ben-Or, and Eban [ABOE10] prove that the answer is essentially "yes"! (The verifier in these proof systems is not quite classical, but nearly so—the best known one uses only a register containing a single qubit [Bro15]). A recent breakthrough by Mahadev [Mah18] shows that under natural cryptographic assumptions, a completely classical verifier can check arbitrary quantum computations!

---

[18]We do not have as yet a satisfactory analog of the PCP Theorem 10.6 (see [AAVL11] for the (subtle) formal definitions and some initial results—far more has happened since).

We stress that this basic, natural idea, of testing scientific theories via interactive procedures, is potentially powerful in other settings as well.

11.4   Quantum randomness: Certification and expansion

One of the most personally satisfying developments in the interaction between computational complexity and quantum mechanics has been the recent flurry of papers on certification and expansion of randomness, which connect several important notions discussed in this book. Let me elaborate. I have given many lectures on pseudo-random generators and on randomness extractors (the topics of Chapters 7 and 9) to varied audiences. Recall that (on top of many theoretical side benefits) these two theories explain how we can salvage the amazing utility of perfect randomness (namely, of independent, unbiased coin flips, which we hypothesize for probabilistic algorithms) in a world that may not have it.[19] In these lectures, the most typical question I get from physics-minded audience members is: "Why bother worrying about such hypothetical worlds? In our world, quantum mechanics suggests simple devices producing a stream of perfect random bits. Don't you trust quantum mechanics?" This is indeed an excellent question, with an excellent answer, related to what we have just discussed above: "Even if I do trust the theory, why should I trust the actual devices?" The basic question here is: How can you test that a given distribution (from which you can only sample) is perfectly random, or has entropy at all?

Let us be more precise about the setting. Suppose that some black box B (operated by your worst enemy) is spewing out a stream of bits, say, n of them. You would like to design a test (i.e. a function) that will output YES/NO and will distinguish the cases where the output is random (say, has positive entropy rate) from the output being deterministic (namely, is a fixed sequence). Clearly, if no assumptions are made about B, this is impossible. By convexity, if any distribution causes an output YES with some probability p, some fixed sequence in the support of this distribution will do at least as well (and our enemy may choose to output this sequence). The same argument remains true if our test is more complex (e.g., we are allowed to feed B some (possibly random) input, on which B's output may depend).

So we need to assume something about the way B operates.[20] The new discovery we relate here is that a most natural physical assumption, indeed far weaker than full-fledged quantum mechanics, suffices. This (classical!) assumption is called *no-signaling*. It assumes that B actually consists of two boxes, B_1 and B_2, which do not communicate in the following strong sense: The output distribution of each is independent of the input to the other[21]. Enforcing no-signaling between devices is in principle possible by appropriate spacial separation, and we will assume it.

So we will attempt to extract *certified* randomness from (say two) no-signaling boxes by means of a game. The key will be the set of strategies these no-signaling boxes can implement. The power of quantum over classical strategies in the no-signaling setting was key to the Bell inequalities and their simplification by [CHSH69] with regard to the EPR paradox and the hidden-variable theory. Indeed, this power is beautifully demonstrated by the original CHSH game [CHSH69], which we now describe. Imagine that a verifier sends independent unbiased bits x_1 and x_2, respectively, to B_1 and B_2, who respond respectively with bits y_1, y_2. Say that the boxes "win" the game if $x_1 \wedge x_2 = y_1 \oplus y_2$, an event the verifier can easily check. What is the maximum probability the boxes win if their strategy must be no-signaling? It is a simple exercise

---

[19]It is quite possible that our world is completely deterministic (providing no randomness at all), or that it has only weak randomness (some entropy is there somewhere, but the "coin flips" we can measure are arbitrarily biased and correlated). But these two respective theories "deal with" such possibilities!

[20]The so-called "no free lunch theorem" says that we must always pay (with assumptions) for what we get. Indeed, for the two theories of randomness above, we postulated (respectively) computational assumptions for pseudo-randomness in a world with no randomness, and postulated the availability of (very few) truly random bits for randomness extractors in a world with "weak" randomness.

[21]A similar definition exists for more than two boxes, which will be needed later. Basically, the joint output distribution of any subset of boxes is independent of the joint inputs to the rest. As an aside, we note that this notion was used in a recent classical result, providing another demonstration of the power of "quantum ideas" in the classical setting. The reader may recall that no communication between *provers* was essential to the multi-prover system $\mathcal{MIP}$ of [BOGKW89] mentioned briefly in Chapter 10 as a precursor to PCPs. The new powerful PCP theorem of [KRR14] for no-signaling provers allows ultra-fast, trusted *delegation* of computation to powerful entities. Other examples of the power of "quantum ideas" in classical settings can be found in the survey [DdW09].

to check that if their strategies are classical (namely, each outputs a probability distribution depending on its input), then the optimal strategy yields a win with probability .75. In contrast, no-signaling is such a weak requirement that it affords the boxes a simple joint strategy (that the reader is invited to find) that wins with probability 1. What is striking is that it is not too hard to design quantum strategies (namely, when the boxes share an entangled pair of qubits), easily implementable by very simple quantum devices, that allow the boxes to win with probability $\cos^2(\pi/8) \approx .853$ (which incidentally happens to be optimal for quantum strategies by the Tsirelson bound [Tsi93]). This gap between classical and quantum power arising from the Bell inequalities (as in particular this game reveals) served for decades as a demonstration of how counterintuitive (and potentially incomplete) quantum mechanics is, how the local hidden variable theory fails to explain it, how the non-commutative probability theory arising from quantum mechanics differs from the commutative one of classical mechanics, etc., etc.—all central to our understanding of this fascinating world in which we actually live.

And then Colbeck, in his 2006 PhD thesis [Col06], found another fundamental aspect that such a classical-quantum gap as this game demonstrates. He made the following observation. If you are repeatedly playing the CHSH game against no-signaling boxes, and they are winning consistently with higher probability than 75%, their output *must* contain entropy (beyond what is supplied by their inputs). To see this, note that otherwise, they are using deterministic strategies, which are in particular classical. So the boxes don't have to be trusted: We can test the statistics of winning over a large number of experiments, and if (say) it exceeds 80%, we accept them as producing randomness (otherwise declaring them faulty). In short, randomness can be certified. This is one remarkable insight!

Now you might start complaining that we wanted a stream of independent, unbiased bits and we are barely getting fractional entropy at best. You might also complain that we have to put in more (and perfect) randomness than we get out, and wonder about the point of the whole exercise. But then, for both issues, you remember the contents of Chapter 9 on randomness extractors and see the light. Indeed, all this was done, in a rapid succession of papers starting with Pironio et al. [PAM+10], showing that a *few* perfect random bits can be expanded into *many*, nearly uniform ones. The latest available results [MS14a, CY14, MS16] achieve the best one can hope for: an unbounded expansion with minimal error. More precisely, a fixed number of no-signaling boxes can *certifiably* generate, with an input seed of k truly random bits, a distribution on any number n of output bits, which is $\exp(-k)$ close to the uniform distribution on n bits.

These constructions and proofs are quite complex, and I only make a few comments about them here. One point is that different parts of the seed will have different functions. One is hiding from the boxes which subset of the many CHSH games played will be used to generate the output. The second is a normal seed to a randomness extractor, used to convert the present entropy in these outputs into a uniform distribution. Another important point is that in this process, the boxes are repeatedly reused for further and further expansion of randomness, using old outputs as new inputs. So it seems like potentially the boxes can use their memory and shared entanglement to generate correlations in the final output. Preventing this is subtle and requires new delicate techniques of quantum information theory. Finally, this new ability of certified randomness has at least one important use in quantum cryptography, namely, for device-independent security of quantum key-distribution [MS14a].

12   Arithmetic complexity

We now leave the Boolean domain and discuss instead the computation of *polynomials* over arbitrary fields. Polynomials, being so basic and so useful, are studied in a variety of mathematical areas. Here we study their complexity: How many arithmetic operations are required to compute natural ones (e.g., the elementary symmetric polynomials, determinant, permanent, matrix multiplication, convolution)? This study is clearly natural from mathematical and practical standpoints. But once the computational complexity machinery of reductions and completeness is applied, it leads to analogs of $\mathcal{P}$ vs. $\mathcal{NP}$ and other complexity questions that seem easier to solve here than in the Boolean world. There are several reasons for this. Computing formal polynomials is strictly stronger than computing the functions they define; there are fewer relations than in the Boolean world (e.g., $x^2 = x$) which restricts the power of arithmetic computation; and finally, more mathematical tools, mainly from algebra, are available. Indeed, progress on arithmetic circuit complexity is faster-paced with exciting new developments, in comparison to Boolean circuit complexity. For this reason, even the recent surveys [SY10, CKW11] are not completely up to date (but do provide detail and proofs for most of the material here and more). Extensive books with scope much wider than we discuss here are [BCS10, vzGG13].

In this chapter, we use the same[1] notation $\mathsf{S}(f)$ to denote the minimal size of an *arithmetic* circuit computing a *polynomial* f. We will formally define it below in general for multivariate polynomials. But for starters, let us discover the difficulty, depth, and unexpected connections of such questions, even for univariate polynomials.

12.1   Motivation: Univariate polynomials

Consider $f(x) \in \mathbb{F}[x]$ of degree d. How many additions and multiplications does it take to compute f (starting from x and any constants from $\mathbb{F}$)? Even this simple question is nontrivial. One clever upper bound is *Horner's rule*, which gives $\mathsf{S}(f) = O(d)$ (try proving it, or peek at the footnote).[2] Another nontrivial fact is an existential lower bound, showing that *some* polynomials require $\mathsf{S}(f) = \Omega(\sqrt{d})$.

But some polynomials are much faster to compute. For example, consider $g(x) = x^d$. Clearly, $\mathsf{S}(g) = O(\log d)$, as we can compute the successive powers $x, x^2, x^4, x^8, \dots$ and then multiply the necessary subset to get x^d. It is also obvious that $\mathsf{S}(g) = \Omega(\log d)$; indeed, at least $\log d$ multiplications are necessary. Amazingly, the following is open.

Open Problem 12.1. Describe a degree d polynomial f for which $\mathsf{S}(f) \neq O(\log d)$.

One natural guess is that the g above is easy, because it has multiple roots. Consider instead the polynomial $h(x) = (x-1)(x-2)\cdots(x-d)$, which has d distinct roots. What is its complexity, over the rationals $\mathbb{Q}$? Besides the obvious $\log d \leq \mathsf{S}(h) \leq d$, nothing is known. But strong upper bounds have an amazing consequence—that factoring integers is easy! Hint: This connection was dubbed "factorials vs. factoring." This and more general results are described by Lipton [Lip94], with some of the ideas dating back to Shamir [Sha79a].

Theorem 12.2 [Sha79a, Lip94]. *If $\mathsf{S}(h) \leq (\log d)^{O(1)}$, then integer factoring is in $\mathcal{P}/\mathrm{poly}$.*

12.2   Basic definitions, questions, and results

We will consider polynomials in $\mathbb{F}[x_1, x_2, \dots]$. The definitions and most questions are interesting over any field $\mathbb{F}$. However, some of the results hold only for certain fields, which for this theory are "large enough," or more specifically, have characteristic zero or are algebraically closed. On the other hand, almost all polynomials we'll discuss will have $0/1$-coefficients, so they make sense over any field. For concreteness, the reader may suppose that $\mathbb{F} = \mathbb{C}$ or $\mathbb{F} = \mathbb{Q}$.

---

[1] As for Boolean circuits.

[2] Hint: Any degree d polynomial can be written as $a_0 + x(a_1 + x(a_2 + x(\cdots + x(a_d))))$, involving d additions and d multiplications.

There are many parallels and differences between Boolean and algebraic circuit complexity, and you may want to recall Section 5.2. We start by defining size.

Just as in Boolean circuits, an *arithmetic circuit* (over $\mathbb{F}$) is a directed acyclic graph in which the non-input gates are labeled with the arithmetic operations $+$ or $\times$. Namely, each gate outputs the polynomial that is, respectively, the sum or product of its two input polynomials. The input nodes can be labeled with the variables x_i, as well as with any constants from $\mathbb{F}$. Thus, for example, the polynomial $\pi x + y$ can be computed with a circuit having 3 inputs, one multiplication gate, and one addition gate. The *size* of a circuit is simply the number of wires in its graph. An *arithmetic formula* is a circuit whose graph is a tree. Allowing the use of constants explains why we don't need a special gate for subtraction. We will soon discuss division as well.

Clearly, every polynomial $f \in \mathbb{F}[x_1, x_2, \dots]$ can be computed by a circuit (and a formula). We denote by $\mathsf{S}(f)$ the minimal size of a circuit for f, and by $\mathsf{L}(f)$ the minimal formula size. We clearly have $\mathsf{S}(f) \leq \mathsf{L}(f)$. Note that we view arithmetic circuits as computing *formal* polynomials, as opposed to the functions they define. This is a distinction over finite fields. For example, the formal polynomials x and x^p are distinct, despite being equivalent as functions over $\mathbb{F}_p$, and so while the size complexity of the first is constant, the second requires size $\geq \log p$.

Like Boolean circuit complexity, arithmetic circuit complexity is an asymptotic theory. We will typically have a sequence $f = \{f_n\}$ of polynomials parameterized by n, which will typically be the number of variables or a polynomially related quantity. For example, the determinant polynomial will be a sequence $DET = \{DET_n\}$, where $DET_n(X)$ is the determinant polynomial in n^2 variables $x_{i,j}$ of the $n \times n$ matrix X. It is important to note that, unlike the Boolean setting, a polynomial has another input parameter on which the complexity may depend, namely, its *degree*. Almost the whole theory deals with multivariate polynomials, and to focus on a single parameter, we will insist that the (total) degree of f_n is at most a fixed polynomial in n as well. In fact, this will hardly be a restriction, as most of our polynomials will be *multilinear* (namely, those in which every variable appears in every monomial with degree at most 1), so the total degree will be automatically bounded by the number of variables.

Arithmetic lower bounds are hard to prove. Indeed, consider even the arithmetic analog of Shannon's Theorem 5.6, which proves the existence of Boolean functions that require exponential size circuits. Recall that it used a counting argument—there were simply more functions than small circuits. However, in arithmetic circuits over an infinite field $\mathbb{F}$, even though the number of "skeleta" of arithmetic circuits is finite, their ability to use arbitrary constants from $\mathbb{F}$ makes their number infinite. Sure, the number of coefficients of monomials of our potentially hard polynomials can also be chosen in infinitely many ways, but we promised to consider only polynomials with $0/1$ coefficients, of which (bounding the number of variables and degree) there are only finitely many. Nevertheless, Hrubeš and Yehudayoff [HY11] proved the following theorem.

Theorem 12.3 [HY11]. *For every field $\mathbb{F}$, almost all multilinear polynomials f on n variables[3] with $0/1$ coefficients require $\mathsf{S}(f) \geq 2^{n/10}$.*

The proof (which gives a more general result) replaces the counting argument with a dimension argument and appeals to basic algebraic geometry. Indeed, not surprisingly, such tools as Bezout's theorem and algebraic transcendence arguments are crucial in this algebraic setting, and they are used also for the few explicit lower bounds we know. We will discuss these in the next section, but the highlights are explicit multilinear polynomials f, g on n variables requiring $\mathsf{S}(f) \geq n \log n$ and $\mathsf{L}(g) \geq n^2/\log n$. Thus in particular, in the arithmetic setting, we do have (slightly) super-linear circuit size lower bounds for polynomials of degree that grows with the number of variables. Here is one challenging open problem, for which the algebraic-geometric tools above seem insufficient.

Open Problem 12.4. Find explicit *constant-degree n-variate* polynomials f for which $\mathsf{S}(f) \neq O(n)$.

The final basic aspect we touch on is the relative power of circuits and formulas. Recall that in the Boolean setting, we believe that circuits are exponentially stronger than formulas. In arithmetic circuits, they are

---

[3]And so of degree $\leq n$.

much closer in power. An important result of Valiant, Skyum, Berkowitz, and Rackoff [VSBR83] shows that arithmetic circuits are amenable to so-called depth reduction. Namely, every circuit computing a degree-d polynomial can be "squashed" to have only $O(\log d)$ alternations between addition and multiplication gates, without significantly increasing its size. For polynomials we care about (namely, with degree $d = n^{O(1)}$), formula size is at most quasi-polynomial in circuit size. We state only the corollary to formula size, proved earlier by Hyafil [Hya79].

Theorem 12.5 [Hya79]. *Let f be a polynomial of degree d. Then $\mathsf{L}(f) \leq \mathsf{S}(f)^{O(\log d)}$.*

12.3   The complexity of basic polynomials

Let us discuss some basic examples of polynomials and what we know about their complexity.

12.3.1   Symmetric polynomials

One important class of polynomials is the *elementary symmetric* polynomials, defined by

$$\mathrm{SYM}_n^k(x_1, x_2, \ldots, x_n) = \sum_{S \subset [n] : |S| = k} \prod_{i \in S} x_i,$$

for all $0 \leq k \leq n$. What is the best way to compute them? Writing them as "sum of products" will take (e.g., for $k = n/2$) exponential size. It turns out that allowing instead "sum of product of sums," a beautiful observation by Ben-Or [BO85] on the power of polynomial interpolation, results in exponential savings, even for formulas.

Theorem 12.6 [BO85]. *For all n and k, $\mathsf{L}(SYM_n^k) \leq O(n^2)$.*

The simple proof arises from noticing that (multivariate) symmetric polynomials in the n variables x_i are the *coefficients* of the (univariate) polynomial $g(t) = \prod_{i=1}^{n}(t + x_i)$ in a new variable t. The formula thus evaluates g on any $n + 1$ distinct values for t (each is a product of sums), and then uses interpolation (which is a linear combination converting values of a polynomial to its coefficients) to compute the desired coefficient from these evaluations. Note that this works only over sufficiently large fields; indeed, we know that over constant size fields; for example, depth-3 (or, indeed, any bounded depth) circuits for middle symmetric polynomials require exponential size.

Such circuits (or formulas), with three alternations between sums and products, are called $\Sigma\Pi\Sigma$-circuits.[4] As it happens, for such depth-3 formulas, this quadratic upper bound is tight [SW01] over all fields.

Suppose we remove the depth restriction, allowing general formulas or circuits. Can we then hope to compute the symmetric polynomials in linear size? This possibility was ruled out in a beautiful combination of two papers by Strassen and Baur-Strassen [Str73a, BS83], again using basic algebraic geometry that we sketch below. It provides the best explicit circuit lower bound we know, and has not been beaten for 40 years!

Theorem 12.7 [Str73a, BS83]. *For all n, $\mathsf{S}(SYM_n^{n/2}) = \Omega(n \log n)$.*

The ideas of this proof are more easily explained for a much simpler family of symmetric polynomials, the *traces* (or *power sums*), so we turn to discuss them. Let

$$T_n^d(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{n} x_i^d.$$

The same authors proved the following.

Theorem 12.8 [Str73a, BS83]. *For all n, d, $\mathsf{S}(T_n^{d+1}) = \Omega(n \log d)$.*

---

[4]The usual representation of a polynomial as a sum of monomials is a $\Sigma\Pi$-circuit, and we will discuss the importance of $\Sigma\Pi\Sigma\Pi$-circuits in Section 12.5.3 below.

As we turn to explain the ideas of this proof in some detail, some readers may want to skip ahead at first reading, and proceed to the next topic of matrix multiplication (Section 12.3.2).

The proof of this theorem follows immediately by combining Theorems 12.9 and 12.10 below, which highlight different nontrivial ways in which arithmetic circuits can compute more efficiently than might be expected, unveiling power that may explain why lower bounds are hard to prove. We need to extend our notation of circuit size measure in the obvious way to circuits with several outputs, which compute several polynomials, denoting it by $\mathsf{S}(f_1, f_2, \ldots, f_m)$. Theorem 12.9 states that the task of computing a sum of powers is not much easier than the task of computing each of the powers separately, and Theorem 12.10 proves the lower bound for that latter task.

Theorem 12.9. *For all n, d, $\mathsf{S}(x_1^d, x_2^d, \ldots, x_n^d) \leq O(\mathsf{S}(T_n^{d+1}))$.*

Theorem 12.10. *For all n, d, $\mathsf{S}(x_1^d, x_2^d, \ldots, x_n^d) = \Omega(n \log d)$.*

Each of these theorems is a special case of a more general one, and we explain them in turn. For the first, Baur and Strassen [BS83] gave a general reduction (greatly simplified by Moregenstern in [Mor85]), from computing the *gradient*[5] ∇f of a polynomial f to computing the polynomial itself. This is one of the nicest examples where an important algorithm is invented to prove a lower bound. Needless to say, computing the gradient of a multivariate function is a basic subroutine in optimization, when performing any of the many variants of the *gradient descent* algorithm. As it turns out, an even more general theorem was discovered earlier in the statistical learning community by Werbos [Wer74, Wer94]. The underlying algorithm, which in that literature is called *back propagation*, is central to the recent revolution in *Deep Learning*, which we briefly discuss in Section 20.6.3.

Theorem 12.11 [BS83, Wer74]. *For every polynomial f on any number n of variables, $\mathsf{S}(\nabla f) \leq O(\mathsf{S}(f))$.*

This is a pretty surprising theorem, as the most obvious way to compute the first partials is one at a time, resulting in a factor n loss in size. But it turns out that computing these can be combined cleverly in a way that loses only a constant factor in size. Before sketching the proof, note that it implies Theorem 12.9 by noting that $\frac{\partial}{\partial x_i} T_n^{d+1} = x_i^d$.

To give a high-level hint of the proof of Theorem 12.11 (following Morgenstern's beautiful argument [Mor85]), imagine that the gates of the circuit for f compute (in order) the polynomials $g_1, g_2, \ldots, g_s$, where the first n g_i are the variables x_i, and the last one is $g_s = f$. Append to this circuit its "mirror image," namely, gates $h_s, \ldots, h_2, h_1$ that will (using the chain rule for partial derivatives of multivariate functions) compute (in this order) $h_i = \partial f / \partial g_i$, using the children of g_i in the original circuit.

Now let us turn to Theorem 12.10. It looks obvious. After all, computing each output x_i^d requires $\log d$ multiplications, as we saw in Section 12.2, and *surely* these n computations cannot be combined, as they involve different variables. Thus we must pay a factor n times the cost of a single task, giving the required $n \log d$ lower bound. Convinced? Like many arguments containing words like "surely" (or "it is easy to see," etc.), the above argument is false, and the bug is precisely in this arrogant word. We will soon see an example where n different computations on completely disjoint variables *can* be nontrivially combined, with a sublinear size increase of only a factor n^c more than a a single task, with $c < 1$. In other words, sometimes a surprising economy of scale is possible in arithmetic computation, which rules out this type of argument.[6]

So another route must be taken to show that for the problem at hand, no economy of scale is possible. Strassen [Str73b] proved the following general *degree* lower bound.

Theorem 12.12 [Str73b]. *For any set of polynomials $f_1, f_2, \ldots, f_m$ on a set variables $x_1, x_2, \ldots, x_n$, we have*

$$\mathsf{S}(f_1, f_2, \ldots, f_m) \geq \log \deg(f_1, f_2, \ldots, f_m).$$

Here, $\deg(f_1, f_2, \ldots, f_m)$ extends the usual notion of degree of a single polynomial. It denotes the degree of the *algebraic variety* defined by the polynomials $f_1 - z_1, f_2 - z_2, \ldots, f_m - z_m)$ (with the z_i new variables

---

[5]Namely, the vector of first partial derivatives of f: $\nabla f = (\frac{\partial}{\partial x_1} f, \frac{\partial}{\partial x_2} f, \ldots, \frac{\partial}{\partial x_n} f)$.

[6]This question—of achieving economy of scale by combining computations of many independent instances of the same problem—is relevant to any computational model. It is called the *direct sum* question, and is understood only for precious few models.

disjoint from the x_j). We will not define it formally here. But the role it plays in this proof is easy to understand in the analogy of this theorem with the trivial univariate case, which already gave us a lower bound of $\mathsf{S}(g) \geq \log \deg(g)$. This univariate lower bound followed from the basic facts about degree; namely, that (a) addition does not increase the maximum degree of the two polynomials it adds, and (b) multiplication at most doubles that maximum degree. As it happens, both (a) and (b) are also satisfied in the multivariate case with that notion of degree; this is guaranteed by a basic algebraic geometric fact called "Bézout's theorem." Using this, Strassen's multivariate theorem follows as the univariate one. To conclude from it that Theorem 12.10 holds it suffices to check that indeed $\deg(f_1, f_2, \ldots, f_m) \geq d^n$, which turns out to follow from the simple fact that the system of polynomial equations $\{f_i = 1\}$ has d^n solutions in $\mathbb{C}$.

12.3.2   Matrix multiplication

Consider next *matrix multiplication* MM, where $\mathrm{MM}_n(X, Y)$ takes two $n \times n$ matrices X, Y and outputs their product XY. Formally, this is a polynomial *map*, as here we compute the n^2 entries of the product, and we consider circuits with so many outputs. The obvious algorithm, performing each inner product separately, gives $\mathsf{S}(\mathrm{MM}) = O(n^3)$, which was considered the best possible for centuries. Strassen [Str69] shocked the mathematics community by proving a sub-cubic bound, $\mathsf{S}(\mathrm{MM}) = O(n^{\log_2 7}) = O(n^{2.8074\cdots})$. With hindsight, you can figure it out yourself: Try to devise a method to multiply two 2×2 matrices using only 7 multiplications (and any number of additions). If you succeed, and furthermore you did not use commutativity of the matrix entries, then you can use recursion for multiplying larger matrices (and check that multiplications dominate the total complexity).

One consequence of this sub-cubic algorithm is to the direct sum (economy-of-scale) problem mentioned in the previous subsection. It shows how arithmetic circuits *can* sometimes achieve nontrivial savings when performing independent tasks jointly. To see this, note that the product XY of two $n \times n$ matrices may be viewed as n instances of matrix-vector products, where we fix the matrix X and let the column vectors of Y be the independent variables. For a typical fixed matrix X, the task of multiplying it by one vector takes n^2 operations. Strassen's fast matrix multiplication algorithm magically combines the computation of n independent such tasks, paying a factor far smaller than n in size.

This algorithmic breakthrough generated a very long sequence of improvements to the exponent, and the current record is $\mathsf{S}(\mathrm{MM}) = O(n^{2.3728639\cdots})$. The rich variety of ideas in this history is surveyed in the PhD thesis of Stothers [Sto10]. The obvious questions are: How far down will the exponent drop? Can it get down to 2?

Open Problem 12.13. Prove or disprove: For every $\epsilon > 0$, $\mathsf{S}(\mathrm{MM}) = O(n^{2+\epsilon})$.

The main line of work, responsible for most progress and leading to the current record, comprises variants and extensions of Strassen's *laser method*. But as you might guess from the number of digits in the record exponent shown, that recent progress (which used heavy computer calculations) has diminishing returns. Ambainis et al. [AFG14] formally encapsulated this set of techniques around the laser method, and proved they get stuck at $n^{2.3078}$. Barriers for more general techniques were recently presented in [AW18, CVZ18].

A completely different, ingenious approach to matrix multiplication was suggested by Cohn and Umans [CU03] (and developed further in [CKSU05, CU13, BCC+17]). It shows how upper bounds on the exponent of matrix multiplication (potentially approaching 2) would follow directly from simple properties of (appropriate) finite groups, thus presenting a concrete challenge to group theorists: Do such appropriate groups exist? I will not state here the conditions required from the group, but only note that it involves the sizes of certain subgroups and the largest dimension of its complex irreducible representations. As beautifully explained in the original paper, this approach to matrix multiplication may be viewed as a non-commutative analog of another gem of arithmetic computation: the $n \log n$ size circuit for the convolution of two n-vectors via the FFT on the cyclic group Z_n. In both cases, the required product is reduced to multiplying two elements in the group algebra of an appropriate group. In the Abelian case (of convolution), Fourier transform reduces this at once to several multiplications of constants, as all irreducible representations are of dimension 1. In the non-Abelian case (of matrix multiplication), Fourier transform reduces this to a series of smaller matrix

multiplications; the sizes depend on the dimensions of the irreducible representations, and these are handled recursively.

12.3.3   The determinant

Next we consider what is perhaps the most important polynomial in mathematics, namely, the *determinant* polynomial DET, defined by the familiar formula (where S_n denotes the symmetric group of permutations on $[n]$):

$$DET_n(X) = \sum_{\sigma \in S_n} \text{sgn}(\sigma) \prod_{i=1}^{n} x_{i,\sigma(i)}.$$

Again, here is another polynomial with exponentially many monomials, which mathematicians have known for centuries has an efficient algorithm for computing it: Gaussian[7] elimination gives $\mathsf{S}(DET) = O(n^3)$. Actually, does it? Recall that Gaussian elimination essentially uses *division*, which we do not allow in our arithmetic circuits. Can division help when computing polynomials? Strassen [Str73b] showed that, up to a polynomial blow up in size, it does not. For determinants, there is no loss at all. A beautiful algorithm of Berkowitz [Ber84] actually *reduces* (in a sense we'll discuss soon) the computation of determinant to computing the product of several matrices. Thus, it actually proves a much stronger upper bound on circuit size.

Theorem 12.14. $\mathsf{S}(DET) \leq O(\mathsf{S}(MM) \log n) = O(n^{2.3728639...})$.

We have no super-linear circuit size lower bounds for determinant. How about formulas? The best upper and lower bounds are given below. The quasi-polynomial upper bound is due to Hyafil [Hya79] (which inspired the Berkowitz algorithm). The cubic lower bound is due to Kalorkoti [Kal85], who developed a technique for general formula lower bounds via a transcendence degree argument.[8]

Theorem 12.15.

- $\mathsf{L}(DET) = O(n^{\log n})$.

- $\mathsf{L}(DET) = \Omega(n^3)$.

12.3.4   The permanent

All the polynomials discussed above can be efficiently computable. We end our example with the prototypical hard polynomial, the *permanent*. It is the monotone sibling of the determinant, defined by

$$PER_n(X) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} x_{i,\sigma(i)}.$$

Despite their structural similarity, the determinant and the permanent are worlds apart. The best known way to compute the permanent is via Ryser's formula [Rys63], which significantly saves on the number $n!$ of monomials but is still exponential. Interestingly, it is a $\Sigma\Pi\Sigma$-formula.

Theorem 12.16 [Rys63]. $\mathsf{L}(PER) = O(n^2 2^n)$.

As we will soon see, the most important open problem in algebraic complexity, proving explicit super-polynomial circuit lower bounds, can be asked about the permanent.

Conjecture 12.17. $\mathsf{S}(PER) \neq n^{O(1)}$.

---

[7]It is named after Gauss mainly for the notation he invented. This method for solving simultaneous linear equations was used by Chinese mathematicians 2,000 years ago.

[8]This general technique can yield a near-quadratic lower bound of $n^2/\log n$ for an explicit multilinear polynomial in n variables (this is a stronger result, since DET_n has n^2 variables). Proving a superquadratic lower bound for such polynomials is an important challenge!

12.4   Reductions and completeness, $\mathcal{VP}$ and $\mathcal{VNP}$

Valiant's paper [Val79a] transformed arithmetic complexity into a complexity theory. In it, he provides the analogs of all basic foundations of Boolean computational complexity:

- Gives a mathematically elegant notion of efficient reducibility between polynomials: *projection*.

- Defines arithmetic analogs of $\mathcal{P}$ and $\mathcal{NP}$, now called, respectively, "$\mathcal{VP}$" and "$\mathcal{VNP}$."

- Endows these classes with natural complete polynomials under such reductions: *permanent* is complete for $\mathcal{VNP}$ and *determinant* is (nearly) complete for $\mathcal{VP}$.

Let us consider all these in turn, and then discuss some consequences. Note that all definitions are *nonuniform*, as for Boolean circuits.

We now define two, nearly equivalent, notions of reduction, Valiant's *projection* and the slightly more general, mathematically standard, *affine projection*.[9] While we mainly use the latter, essentially all results hold for both.

Definition 12.18 (Projection and affine projection)**.** Let $f \in \mathbb{F}[x_1, x_2, \ldots, x_n]$ and $g \in \mathbb{F}[y_1, y_2, \ldots, y_m]$. We say that f is an *affine projection* of g, written $f \leq g$, if there exist m affine functions $\ell_i : \mathbb{F}^n \to \mathbb{F}$ such that $f(x) = g(\ell_1(x), \ell_2(x), \ldots, \ell_m(x))$. We say that f is a *projection* of g if it is an affine projection where all affine functions ℓ_i depend on at most one variable.

It is clear that these reductions are efficient in terms of circuit size; if $f \leq g$ then $\mathsf{S}(f) \leq \mathsf{S}(g) + O(mn)$, as given a circuit for g, we can feed its inputs the affine functions ℓ_i to get a circuit for f. This relation is clearly transitive, and so gives a partial order on the relative complexity of polynomials, as in the Boolean world.

The class $\mathcal{VP}$, in complete analogy to $\mathcal{P}/\text{poly}$, is simply all polynomials computable by polynomial-size arithmetic circuits. Remember that for all polynomials discussed, the degree is polynomially bounded by the number of variables. Thus, for example, the polynomials SYM, MM, DET of the previous section are all in $\mathcal{VP}$.

Definition 12.19 (The class $\mathcal{VP}$)**.** We say that $f = \{f_n\}$ is in $\mathcal{VP}$ if $\mathsf{S}(f) \leq n^{O(1)}$.

Defining the analog $\mathcal{VNP}$ of $\mathcal{NP}$ is a bit more complicated, but nevertheless natural. In $\mathcal{NP}$ an existential quantifier is used, which can be viewed as a Boolean *disjunction* over all possible Boolean values to potential witnesses, or "certificates," in a polynomial-size Boolean circuit. In the arithmetic $\mathcal{VNP}$, this disjunction is replaced by a *summation* over possible witnesses in a polynomial-size arithmetic circuit (thus effectively converting computing the existence of certain objects into counting them). It is a nontrivial and important choice to still take this sum only over *Boolean* values, regardless of the underlying field.

Definition 12.20 (The class $\mathcal{VNP}$)**.** We say that $f = \{f_n\} \in \mathbb{F}[x_1, x_2, \ldots, x_n]$ is in $\mathcal{VNP}$ if there exists $g = \{g_n\} \in \mathbb{F}[x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_n] \in \mathcal{VP}$ such that $f_n(x)$ is defined from $g_n(x, y)$ via $f_n(x) = \sum_{\alpha \in \{0,1\}^n} g_n(x, \alpha)$.

We clearly have $\mathcal{VP} \subseteq \mathcal{VNP}$, and the major problem of arithmetic complexity theory is proving the following conjecture.

Conjecture 12.21. $\mathcal{VP} \neq \mathcal{VNP}$.

Finally, we get to the complete problems, determinant and permanent. Noting that these two polynomials are identical when the field $\mathbb{F}$ has characteristic 2, we now consider only fields with characteristic different than 2. Valiant proved two completeness theorems. The one for $\mathcal{VNP}$ is completely clean.

---

[9]Note that most polynomial-time reductions discussed for Boolean computation are actually projections, or a very simple Boolean function of a few projections.

Theorem 12.22 [Val79a]. *PER is $\mathcal{VNP}$-hard.*
More precisely, for every $f = \{f_n\} \in \mathcal{VNP}$ and every n, $f_n \leq PER_{n^{O(1)}}$.

The reader is invited to verify that also $PER \in \mathcal{VNP}$, so the theorem implies that PER is actually a complete polynomial for $\mathcal{VNP}$. The completeness of the determinant is a bit harder to state. First, we need to define an important subclass of $\mathcal{VP}$, which is called $\mathcal{VL}$, of all polynomials that have polynomial-size formulas.

Theorem 12.23 [Val79a]. *DET is $\mathcal{VL}$-hard. More precisely, for every $f = \{f_n\} \in \mathcal{VL}$ and every n, $f_n \leq DET_{n^{O(1)}}$.*

As DET is in $\mathcal{VP}$, is hard for $\mathcal{VL}$, and by Theorem 12.5, these two classes are nearly equal (up to quasi-polynomial factors), we get a precise meaning for DET-completeness in $\mathcal{VP}$. Namely, every polynomial $f \in \mathcal{VP}$ is a projection of an $n^{O(\log n)}$ determinant.

These completeness results, besides providing a starting point for many other reductions (as was the role SAT played in Boolean complexity), highlight the importance of these two polynomials, permanent and determinant, and may partly explain the important role they play in mathematics. First, the determinant appears everywhere in mathematics—lots of useful polynomials that naturally arise are expressible as determinants (e.g., *Jacobians* in calculus, *Alexander polynomials* in knot theory, *Wronskians* in differential equations, *characteristic polynomials* and *resultants* in algebra, *volumes* of parallelepipeds in geometry, and numerous others). This phenomenon may be less surprising, now that we know that every polynomial that can be described by a small formula has an equally small determinantal representation. Similarly, the permanent appears quite frequently as well. It turns out to capture the *Tutte* and *chromatic polynomials* in graph theory, the *Jones polynomials* in knot theory, and many *partition functions* arising in statistical mechanics models; it also counts integer points in convex sets, counts extensions of partially ordered sets, and so forth. Unlike the examples for determinants, these seem hard to compute.

Another important contribution of these completeness results is that the major problem of separating $\mathcal{VP}$ from $\mathcal{VNP}$ can be cast as a question about the best projections from PER to DET. More precisely, let $m(n)$ be the smallest integer for which $PER_n \leq DET_{m(n)}$. Then the completeness results imply the following.

Corollary 12.24. *If $\mathcal{VP} \neq \mathcal{VNP}$ then $m(n) \neq n^{O(1)}$. And almost conversely, if $m(n) \neq n^{O(\log n)}$, then $\mathcal{VP} \neq \mathcal{VNP}$.*

Attempts to study $m(n)$ started with Polya, who noticed[10] that $m(2) = 2$. It may not be even clear that $m(n)$ is always finite, but of course combining Theorems 12.16 and 12.23, we get that $m(n) \leq \exp(n)$. The best known lower bound, due to Mignon and Ressayre [MR04] (extended by [CCL10] to all fields), is quadratic.

Theorem 12.25. $m(n) \geq \Omega(n^2)$.

The proof is essentially linear-algebraic, and it uses only simple properties of linear projection and of the Hessian of the determinant polynomial. Improving this quadratic bound is a major challenge.

Separating $\mathcal{VP}$ from $\mathcal{VNP}$

An ambitious program to prove super-polynomial or even exponential lower bounds on $m(n)$ and separate $\mathcal{VP}$ from $\mathcal{VNP}$ was suggested by Mulmuley and Sohoni (see the surveys [BLMW11, Mul11, Mul12a, Gro15], and our discussion in Section 13.9.1). It crucially uses the fact that both the permanent and determinant polynomials are determined by their symmetries. These symmetries are subgroups of the linear groups, and the affine projection reduction is linear as well. This allows formalizing the problem of $\mathcal{VP}$ vs. $\mathcal{VNP}$ as a question about the intersection of the algebraic varieties defined by the orbit closures (under their natural symmetry groups) of the determinant and permanent polynomials. This formulation naturally suggests using tools from invariant theory, representation theory, and algebraic geometry (some more details are provided in Section 13.9). There seem to be severe obstacles to this program so far (see, e.g., [BIP16]), but this focus

---

[10]Via the simple projection PER $\left(\begin{smallmatrix} x & y \\ z & w \end{smallmatrix}\right) = $ DET $\left(\begin{smallmatrix} x & -y \\ z & w \end{smallmatrix}\right)$.

on using symmetry, and the tools developed, may serve to elucidate (and perhaps prove new lower bounds for) other problems in arithmetic complexity.

Another, completely different and counterintuitive route to separating $\mathcal{VP}$ vs. $\mathcal{VNP}$ is through the design of an efficient deterministic algorithm for a certain problem in Boolean complexity, about arithmetic complexity. This direction is a major bridge between these two fields, and ties the two to pseudo-randomness and de-randomization. The problem at hand is the *polynomial identity testing* problem (or PIT for short). It asks whether a given arithmetic formula computes the identically zero polynomial. Almost equivalently (due to Theorem 12.23), in its original form asked by Edmonds [Edm67b], the same question asks whether the determinant of a given *symbolic matrix* (whose entries are linear forms in some variables), vanishes identically. This problem has a simple efficient probabilistic algorithm (discussed at the beginning of Section 7.1), but the best known deterministic algorithm for it requires exponential time. Surprisingly, Kabanets and Impagliazzo [KI04] proved that significant improvement of the deterministic complexity (namely, nontrivial de-randomization) will entail explicit lower bounds on either arithmetic or Boolean complexity. More on the PIT problem, and partial progress on it and its relatives, can be found in Section 4 of the survey [SY10]. More recent progress, from a very different direction can be found in [GGOW15].

Finally, as in Boolean complexity (see Chapter 5), one can and should wonder about *why* we've been stuck also in the arithmetic setting for decades, unable to prove strong lower bounds and separate $\mathcal{VP}$ from $\mathcal{VNP}$. Indeed, it seems that the arithmetic setting is more limited (or structured) than the Boolean one, and we also potentially have many more tools from mathematics to apply in lower bounds. To explain this, as in the Boolean setting, we need *barrier* results, showing why current techniques have so far had limited success. These have only recently been developed. Conditional barriers, similar in nature to Natural Proofs (see Section 5.2.4), were developed in [FSV17, GKSS17]. An unconditional barrier, for the linear algebraic methods capturing most existing lower bounds in arithmetic complexity, was developed in [EGOW17].

12.5   Restricted models

As for Boolean circuits, our inability to prove strong lower bounds for the general model invites the study of restricted ones, often of interest in their own right. We describe some of them where super-polynomial lower bounds are known, but some substantial challenges remain despite their restricted nature. A similar effort, which was described in less detail in previous chapters, is being made for Boolean circuits, proof complexity, and a variety of other computational settings. In all, we are (hopefully) inching our way into truly general lower bounds through the development of new techniques, and are testing our mettle by proving lower bounds on a variety of restricted computational models.

12.5.1   Monotone circuits

Monotone circuits make sense for ordered fields like $\mathbb{R}$ or $\mathbb{Q}$. They are defined just as general circuits are except that they can only use *positive* coefficients from the field. Monotone circuits can clearly compute any polynomial with positive coefficients. Just as in the Boolean case, we have exponential lower bounds as well as natural separations between monotone and nonmonotone circuits. Indeed, these arithmetic bounds predate the Boolean ones and are significantly simpler.

Theorem 12.26 [SS79, TT94]. *PER requires monotone circuits of size* $\exp(n)$.

Theorem 12.27 [Val80]. *There is a positive polynomial* $f \in \mathcal{VP}$ *that requires* $\exp(n)$ *monotone circuits.*

12.5.2   Multilinear circuits

In multilinear circuits and formulas, every gate must compute a multilinear function. Clearly, such circuits can compute every multilinear polynomial.

Theorem 12.28 [Raz04a]. *DET requires multilinear formulas of size* $n^{\Omega(\log n)}$.

We know $\mathsf{L}(DET) \leq n^{O(\log n)}$, but the formula supplying this upper bound is *not* multilinear. Indeed, it is believed that DET actually requires exponential size multilinear circuits. Proving this, as well as proving

any super-polynomial circuit size lower bounds for multilinear circuits are important open problems.

Tensors A natural sub-family of multilinear functions is defined by *tensors*. Tensors extend matrices in a natural way—a d-tensor is described by a d-dimensional array of coefficients of the associated d-form (in the same way a matrix defines a bi-linear form). Understanding tensors is central to numerous areas of math, physics and computer science. An extensive treatment of the geometry, complexity and applications of tensors is Lansberg's book [Lan12].

The most basic complexity measure of a tensor is its *rank*; again, extending that notion from matrices, for a given d-tensor T, its rank $\mathrm{rk}(T)$ is the minimum number of *rank-1* tensors which sum to T. As rank-1 tensors are simply products of d linear forms, the computational model defined by rank is extremely simple: an arithmetic $\Sigma\Pi\Sigma$-circuit (sum of products of sums), discussed more in the next subsection. Can we prove lower bounds for tensor rank then? This cannot be easy. Sadly, unlike the case of matrices (d=2), where rank is completely understood and efficiently computable, for $d > 2$ the structure is far more complicated, and computing rank becomes $\mathcal{NP}$-complete [Hås90].

Let us relate what is known for the simplest non-trivial case, $d = 3$. Determining the rank of 3-tensors is essential in many applications; for example, the matrix multiplication exponent discussed above is determined by the natural 3-dimensional matrix multiplication tensor. Counting arguments show that (over any field) most $n \times n \times n$ tensors have rank $\Omega(n^2)$. But the best lower bound [AFT11] for any explicit 3-tensor is only $3n$. The difficulty is partly explained by recent barrier results [EGOW17]. They show that most methods used so far for such lower bounds (including *flattenings* used by algebraic geometers) cannot yield better than $6n$ rank lower bounds. This raises the following natural challenge, of obtaining explicit super-linear lower bounds.

Open Problem 12.29. Describe an explicit family T_n of 3-tensors (of size n), with $\mathrm{rk}(T_n) \neq O(n)$.

12.5.3 Bounded-depth circuits

This model sounds suspiciously restricted. Let us clarify its importance, and what exciting progress has been made very recently in arithmetic complexity by studying it.

Bounding the number of alternations of operations is standard in Boolean complexity and logic (e.g., first-order theories allow a finite number of alternations between existential and universal quantifications). We have also already seen in Section 12.2 similar restrictions on the number of alternations between addition and multiplication of Boolean circuits. For example, $\Sigma\Pi$-circuits captured the standard way of writing polynomials, as a sum of monomials. We saw in Theorem 12.6 that allowing one more alternation, $\Sigma\Pi\Sigma$-circuits (sum of products of sums), can give exponential advantage (e.g., in computing the symmetric polynomials). And it stands to reason that allowing one more alternation will be more powerful, and so forth. However, it was taken for granted that the decades-long study of such restricted circuits was mainly to slowly develop tools for "the real thing," general circuit lower bounds. This sentiment changed overnight with a paper by Agrawal and Vinay [AV08]. They realized that the ideas of depth-reduction (Theorem 12.5) can be pushed to squash circuits nontrivially to only four alternations, namely, to $\Sigma\Pi\Sigma\Pi$-circuits. Their result was sharpened by both Koiran and Tavenas [Koi12, Tav13] to produce the following.

Theorem 12.30. *If $f \in \mathcal{VP}$, then f has a $\Sigma\Pi\Sigma\Pi$-circuit of size $n^{O(\sqrt{n})}$. Moreover, if f is homogeneous, the resulting circuit is homogeneous.*[11]

Thus, to prove general lower bounds, "all" we need are lower bounds for homogeneous $\Sigma\Pi\Sigma\Pi$-circuits. This gave a huge energy boost to attacks on such circuits, with the breakthrough result of Gupta, Kamath, Kayal, and Saptharishi [GKKS13], which came extremely close to proving such lower bounds. A sequence of refinements led to a matching lower bound by Saraf and Kumar [SK14].

Theorem 12.31. *There exists an explicit homogeneous polynomial $f \in \mathcal{VP}$, which requires $\Sigma\Pi\Sigma\Pi$-circuits of size $n^{\Omega(\sqrt{n})}$.*

---

[11]Namely, every gate in the circuit computes a homogeneous polynomial.

Note that by the previous theorem, *any* nonconstant improvement to the exponent in this lower bound (e.g., for the permanent) would separate $\mathcal{VP}$ and $\mathcal{VNP}$! The lower bound of this theorem is achieved through the use of *projected, shifted partial derivatives* initiated by [GKKS13]. I will not explain this technique but will briefly explain its progenitor, the *partial derivatives* technique. It was introduced by Nisan and Wigderson [NW96], who used it to prove lower bounds for the weaker $\Sigma\Pi\Sigma$-circuits and other restricted models. These and more applications of the partial derivative method are surveyed in [SY10, CKW11]. The main idea is to define the following *complexity measure* on polynomials. For a multivariate polynomial g, consider the set $PD(g)$ to be all polynomials that are partial derivatives of g (of all orders), and let $\dim(g)$ demote the dimension of the linear span of $PD(g)$. This measure is small for input variables, and it turns out to "progress slowly" under addition and multiplication. Thus, polynomials f with large $\dim(f)$ require large circuits!

12.5.4   Non-commutative circuits

Non-commutativity is prevalent not only in math, but in life as well. Indeed, most pairs of actions we encounter or consider do not commute. Non-commutative polynomials occur naturally when the variables take values in a non-commutative ring, such as rings of matrices, or group algebras of non-commutative groups.

So far in this section, we have implicitly assumed commutativity (namely, that all our variables x_i pairwise commute). Now let us drop this assumption and discuss non-commutative polynomials, and circuits and formulas for them. For non-commutative polynomials, one has to specify the order of variable appearance in every monomial (e.g., xy and yx are different monomials). Similarly, in circuits and formulas, we must specify the order of multiplication in product gates. A good demonstration of the weakness of this model is that while in the commutative setting we can compute $x^2 - y^2$ using *one* multiplication only, as $(x-y)(x+y)$, this is impossible in the non-commutative setting.

Nisan [Nis91a] proved exponential formula lower bounds for determinant[12] (and permanent), and an exponential gap between the power of formulas and that of circuits.

Theorem 12.32 [Nis91a]. *PER and DET require non-commutative formulas of size* $\exp(n)$.

Theorem 12.33 [Nis91a]. *There is a non-commutative polynomial with a linear size non-commutative circuit, which requires* $\exp(n)$ *non-commutative formulas.*

Note that the last theorem means that the depth reduction of Theorem 12.5, showing that formulas and circuits have near-equal power in the commutative case, is false in the non-commutative setting. Indeed, there are many other differences between these two worlds. A surprising one, due to Arvind and Srinivasan [AS10], is that in the non-commutative setting, the (Cayley versions of) permanent and determinant are equally hard: $DET \leq PER$ and $PER \leq DET$! Yet another issue we discussed, Strassen's efficient elimination of division gates when computing polynomials, is not known to be possible in the non-commutative setting. This is studied in [HW14], and is very closely related to certain problems in invariant theory discussed in Section 13.9.3.

The central problem in this area is proving super-polynomial circuit lower bounds. One attack [HWY10] shows how such (even exponential) lower bounds can be deduced from certain super-linear *commutative* circuit lower bounds.

---

[12]As discussed, to formally define this polynomial, one has to order the variables in each monomial. A natural ordering we pick is row-order, called the *Cayley determinant*.

13 *Interlude*: Concrete interactions between math and computational complexity

The introduction discussed the variety of interactions between math and computation at a high level. In this chapter we will meet concrete examples of interactions of computational complexity theory with different fields of mathematics. I aim for variety—this section demonstrates that hardly any area of modern mathematics is untouched by this computational connection, which in some cases is quite surprising.

I have chosen to focus on essentially one problem or development in each mathematical field. Typically, this touches only a small subarea, which does not do justice to a wealth of connections. Thus, each section should be viewed as a demonstration of a larger body of work and even bigger potential. Indeed, while in some areas the collaborations are reasonably well established, in others they are just budding, with lots of exciting problems waiting to be solved and theories to be developed. The descriptions are relatively short, but they include background and intuition, as well as further reading material. The vignettes in this chapter will hopefully tempt the reader to explore deeper.

The selection of fields and foci is affected by my personal taste and limited knowledge. Connections to other fields, like combinatorics, optimization, logic, topology, and information theory already appear in parts of this text. Undoubtedly others could be added. Here is the list of the areas covered and topics chosen in each; these sections can be read in any order:

- Number theory: *Primality testing*

- Combinatorial geometry: *Point-line incidence*

- Operator theory: *The Kadison-Singer problem*

- Metric geometry: *Distortion of embeddings*

- Group theory: *Generation and random generation*

- Statistical physics: *Monte-Carlo Markov chains*

- Analysis and probability: *Noise stability*

- Lattice theory: *Short vectors*

- Invariant theory: *Actions on matrix tuples*

13.1 Number theory

As mentioned, the need to efficiently compute mathematical objects has been central to mathematicians and scientists throughout history, and of course, the earliest subject is arithmetic. Perhaps the most radical demonstration of this focus is the place value system we use to represent integers, which has been used for millennia precisely because it supports extremely efficient manipulation of arithmetic operations. The next computational challenge in arithmetic, since antiquity, was accessing the multiplicative structure of integers represented this way.

Here is an except from C. F. Gauss' appeal[1] to the mathematics community of his time (in article 329 of *Disquisitiones Arithmeticae* (1801)), regarding the computational complexity of testing primality and factoring integers into their prime components. The importance that Gauss assigns to this computational challenge, his frustration with the state of the art, and his imploring the mathematical community to resolve it all shine through:

---

[1]Which is of course in Latin. I copied this English translation from a wonderful survey by Granville [Gra05] on the subject matter of this section.

The problem of distinguishing prime numbers from composite numbers, and of resolving the latter into their prime factors is known to be one of the most important and useful in arithmetic. It has engaged the industry and wisdom of ancient and modern geometers to such an extent that it would be superfluous to discuss the problem at length. Nevertheless we must confess that all methods that have been proposed thus far are either restricted to very special cases or are so laborious and difficult that even for numbers that do not exceed the limits of tables constructed by estimable men, they try the patience of even the practiced calculator. And these methods do not apply at all to larger numbers ... the dignity of the science itself seems to require that every possible means be explored for the solution of a problem so elegant and so celebrated.

I briefly recount the state of the art of these two basic algorithmic problems in number theory. A remarkable response to Gauss' first question, *efficiently deciding primality*, was found in 2002 by Agrawal, Kayal, and Saxena [AKS04]. The use of symbolic polynomials for this problem is completely novel. Here is their elegant characterization of prime numbers.

Theorem 13.1 [AKS04]. *An integer $N \geq 2$ is prime if and only if*

- *N is not a perfect power,*

- *N does not have any prime factor $\leq (\log N)^4$, and*

- *For every $r, a < (\log N)^4$, we have the following equivalence of polynomials over $\mathbb{Z}_N[X]$:*

$$(X + a)^N \equiv X^N + a \mod (X^r - 1).$$

It is not hard to see that this characterization gives rise to a simple algorithm for testing primality that is deterministic, and runs in time that is polynomial in the binary description length of N. Previous deterministic algorithms either assumed the generalized Riemann hypothesis [Mil76] or required slightly super-polynomial time [APR83]. The AKS deterministic algorithm came after a sequence of efficient probabilistic algorithms [SS77, Rab80, GK86, AH92], some elementary and some requiring sophisticated use and development of number-theoretic techniques. These probabilistic and deterministic algorithms were partly motivated by, and are important to, the field of cryptography.

What is not so well known, even for those who did read the beautiful, ingenious proof in [AKS04], is that AKS developed their deterministic algorithm by carefully de-randomizing a previous probabilistic algorithm for primality of [AB03] (which uses polynomials). Note that *de-randomization*, the conversion of probabilistic algorithms into deterministic ones, is now a major area in computational complexity with a rich theory and many other similar successes as well as challenges. The stunning possibility that *every* efficient probabilistic algorithm has a deterministic counterpart is one of the major problems of computational complexity, and there is strong evidence supporting it (see [IW97]). Much more on this can be found in the chapters dealing with randomness, especially Chapter 7.

Gauss' second challenge, of whether efficiently factoring integers is possible, remains open. But this very challenge has enriched computer science, both practical and theoretical, in several major ways. Indeed, the assumed hardness of factoring is the main guarantee of security in almost all cryptographic and e-commerce systems around the world (showing that difficult problems can be useful!). More generally, cryptography is an avid consumer of number-theoretic notions, including elliptic curves, Weil pairings, and more, which are critical to a variety of cryptographic primitives and applications. These developments shatter Hardy's view of number theory as a completely useless intellectual endeavor.

There are several problems on integers whose natural definitions depend on factorization but can nevertheless be solved efficiently, bypassing the seeming need to factor. Perhaps the earliest algorithm ever formally described is Euclid's algorithm for computing the greatest common divisor of two given integers[2] m and n. Another famous such algorithm is for computing the Legendre-Jacobi symbol $\left(\frac{m}{n}\right)$ via Gauss' law of quadratic reciprocity.

---

[2]It extends to polynomials and allows an efficient way of computing multiplicative inverses in quotient rings of $\mathbb{Z}$ and $\mathbb{F}[x]$.

A fast algorithm for factoring may come out of left field with the new development of quantum computing, the study of computers based on quantum-mechanical principles, which we discussed in Chapter 11. Shor has shown in [Sho94] that such computers are capable of factoring integers in polynomial time. This result led governments, companies, and academia to invest billions in developing technologies that will enable building large-scale quantum computers, and the jury is still out on the feasibility of this project. There is no known theoretical impediment for doing so, but one possible reason for the failure of this project to date is the existence of yet-undiscovered principles of quantum mechanics.

Other central computational problems include solving polynomial equations in finite fields, for which one of the earliest efficient (probabilistic) algorithms was developed by Berlekamp [Ber67] (it remains a great challenge to de-randomize this algorithm). Many other examples can be found in the algorithmic number theory book [BS97].

13.2 Combinatorial geometry

What is the smallest area of a planar region that contains a unit-length segment in *every* direction? This is the Kakeya needle problem (and such sets are called *Kakeya sets*), which was solved surprisingly by Besicavitch [Bes19], who showed that this area can be arbitrarily close to zero! A slight variation on his method produces a Kakeya set of Lebesgue measure zero. It makes sense to replace "area" (namely, Lesbegue measure) by the more robust measures, such as the Hausdorff and Minkowski dimensions. This changes the picture: Davies [Dav71] proved that a Kakeya set in the plane must have full dimension (=2) in both measures, despite being so sparse in Lebesgue measure.

It is natural to extend this problem to higher dimensions. However, obtaining analogous results (namely, that the Hausdorff and Minkowski dimensions are full) turns out to be extremely difficult. Despite the seemingly recreational flavor, this problem has significant importance in a number of mathematical areas (Fourier analysis, wave equations, analytic number theory, and randomness extraction) and has been attacked through a considerable diversity of mathematical ideas (see [Tao09]).

The following finite field analog of the above Euclidean problem was suggested by Wolff [Wol99]. Let $\mathbb{F}$ denote a finite field of size q. A set $K \subseteq \mathbb{F}^n$ is called Kakeya if it contains a line in every direction. More precisely, for every direction $b \in \mathbb{F}^n$, there is a point $a \in \mathbb{F}^n$ such that the line $\{a + bt : t \in \mathbb{F}\}$ is contained in K. As above, we would like to show that any such K must be large (think of the dimension n as a large constant, and the field size q as going to infinity).

Conjecture 13.2. Let $K \subseteq \mathbb{F}^n$ be a Kakeya set. Then $|K| \geq C_n q^n$, where C_n is a constant depending only on the dimension n.

The best exponent of q in such a lower bound intuitively corresponds to the Hausdorff and Minkowski dimensions in the Euclidean setting. Using sophisticated techniques from arithmetic combinatorics, Bourgain, Tao, and others improved the trivial bound of $n/2$ to about $4n/7$.

Curiously, the same conjecture arose, completely independently, in ToC, from the work [LRVW03] on *randomness extractors*, an area that studies the purification of weak random sources, which we discussed in Chapter 9 (see e.g., the survey [Vad11]). With this motivation, Dvir [Dvi09] brilliantly proved the Wolff conjecture (sometimes called the "finite field Kakeya conjecture"), using the (algebraic-geometric) polynomial method (which is inspired by techniques in decoding algebraic error-correcting codes). Many other applications of this technique to other geometric problems quickly followed, including the Guth-Katz [GK10] resolution of the famous Erdős distance problem, as well as for optimal randomness extraction and more (some are listed in Dvir's survey [Dvi10]).

Subsequent work to Dvir's breakthrough [Dvi09] determined the exact value of the constant C_n above (up to a factor of 2) [DKSS13].

Theorem 13.3 [DKSS13]. *Let $K \subseteq \mathbb{F}^n$ be a Kakeya set. Then $|K| \geq (q/2)^n$. However, there exist Kakeya sets of size $\leq 2 \cdot (q/2)^n$.*

Many other problems regarding incidences of points and lines (and higher-dimensional geometric objects) have been the source of much activity and collaboration among geometers, algebraists, combinatorialists,

and computer scientists. The motivation for these questions on the computer science side come from various sources, such as problems on local correction of errors [BDWY13] and de-randomization [DS07, KS09]. Other incidence theorems (e.g., Szemerédi-Trotter [STJ83] and its finite field version of Bourgain-Katz-Tao [BKT04]) have been used, for example, in randomness extraction [BIW06], compressed sensing [GLR10], and expander construction (see Section 8.7).

13.3   Operator theory

The following basic mathematical problem of Kadison and Singer from 1959 [KS59] was intended to formalize a basic question of Dirac concerning the "universality" of measurements in quantum mechanics. We need a few definitions. Consider $B(\mathcal{H})$, the algebra of continuous linear operators on a Hilbert space $\mathcal{H}$. Define a *state* to be a linear functional f on $B(\mathcal{H})$, normalized to $f(I) = 1$, which takes nonnegative values on positive semidefinite operators. The states form a convex set, and a state is called *pure* if it is not a convex combination of other states. Finally, let D be the subalgebra of $B(\mathcal{H})$ consisting of all *diagonal* operators (after fixing some basis).

Kadison and Singer asked whether every pure state on D has a *unique* extension to $B(\mathcal{H})$. This problem on infinite-dimensional operators found a host of equivalent formulations in finite dimensions, with motivations and intuitions from operator theory, discrepancy theory, Banach space theory, signal processing, and probability. All of them were solved affirmatively in the recent work of Marcus, Spielman, and Srivastava [MSS13b] (which also surveys the many related conjectures). Here is one statement they prove, which implies the others.

Theorem 13.4 [MSS13b]. *For every $\epsilon > 0$, there is an integer $k = k(\epsilon)$ such that the following holds. Fix any n and any $n \times n$ matrix A with zeros on the diagonal and of spectral norm 1. Then there is a partition of $\{1, 2, \cdots, n\}$ into k subsets, $S_1, S_2, \cdots, S_k$, so that each of the principal minors A_i (namely, A restricted to rows and columns in S_i) has spectral norm at most ϵ.*

This statement clearly implies that one of the minors has linear size, at least n/k. This consequence is known as the *restricted invertibility* theorem of Bourgain and Tzafriri [BT91], itself an important result in operator theory.

How did computer scientists get interested in this problem? Without getting into too many details, here is a sketchy description of the meandering path that led to this spectacular result.

A central computational problem, at the heart of numerous applications, is solving a linear system of equations. While Gaussian elimination does the job quite efficiently (the number of arithmetic operations is about n^3 for $n \times n$ matrices), for large n this is still inefficient, and faster methods are sought, hopefully nearly linear in the number of nonzero entries of the given matrix. For Laplacian[3] linear systems (arising in many graph theory applications, such as computing electrical flows and random walks), Spielman and Teng [ST11] achieved precisely that. A major notion they introduced was *spectral sparsifiers* of matrices (or weighted graphs).

A sparsifier of a given matrix is another matrix, with far fewer (indeed, linear) nonzero entries, which nevertheless has essentially the same (normalized) spectrum as the original (it is not even obvious that such a sparse matrix exists). Note that a very special case of sparsifiers of complete graphs are by definition *expander graphs*[4] (see much more about this central concept of expanders in [HLW06] and Section 8.7). The algorithmic applications led to a quest for optimal constructions of sparsifiers for arbitrary Laplacian matrices (in terms of trade-off between sparsity and approximation), and these were beautifully achieved in [BSS14], who also provided a deterministic polynomial-time algorithm to construct such sparsifiers. This in turn has led [SS12] to a new proof, with better analysis, of the restricted invertibility theorem mentioned above, making the connection to the Kadison-Singer problem.

However, the solution to Kadison-Singer seemed to require another detour. The same team [MSS13a]

---

[3]Simply, symmetric PSD matrices with zero row sum.

[4]All nontrivial eigenvalues of the complete graph (or constant matrix) are 0, and an expander is a sparse graph in which all nontrivial eigenvalues are tiny.

first resolved a bold conjecture of Bilu and Linial [BL06] on the spectrum of "signings" of matrices.[5] This conjecture was part of a plan for a simple, iterative construction of Ramanujan graphs, the best[6] possible expander graphs. Ramanujan graphs were introduced and constructed in [LPS88, Mar88] but rely on deep results in number theory and algebraic geometry (believed by some to be essential for any such construction). Bilu and Linial sought instead an elementary construction and made progress on their conjecture, showing how their iterative approach gives yet another way to construct "close to" Ramanujan expanders.

To prove the Bilu-Linial conjecture (and indeed produce Ramanujan graphs of every possible degree— something the algebraic constructions couldn't provide), [MSS13a] developed a theory of *interlacing polynomials* that turned out to be the key technical tool for resolving Kadison-Singer in [MSS13b]. In both cases, the novel view is to think of these conjectures probabilistically and analyze the norm of a random operator by analyzing the average characteristic polynomial. That this method makes sense and actually works is deep and mysterious. Moreover, it provides a new kind of existence proof for which no efficient algorithm (even probabilistic) of finding the desired objects is known. The analysis makes heavy use of the theory of *real stable* polynomials, and the inductive process underlying it is reminiscent of (and inspired by) Gurvits' [Gur08] remarkable proof of the van der Waerden conjecture and its generalizations.[7]

13.4   Metric geometry

How close one metric space is to another is captured by the notion of *distortion*, measuring how distorted distances of one become when embedded into the other.

Definition 13.5. Let (X, d) and (X', d') be two metric spaces. An embedding $f : X \to X'$ has distortion $\leq c$ if for every pair of points $x, y \in X$, we have

$$d(x, y) \leq d'(f(x), f(y)) \leq c \cdot d(x, y).$$

When X is finite and of size n, we allow $c = c(n)$ to depend on n.

Understanding the best embeddings between various metric and normed spaces has been a long endeavor in Banach space theory and metric geometry. An example of one major result in this area is Bourgain's embedding theorem [Bou85].

Theorem 13.6 [Bou85]. *Every metric space of size n can be embedded into Euclidean space L_2 with distortion $O(\log n)$.*

The first connection between these structural questions and computational complexity was made in the important paper of Linial, London, and Rabinovich [LLR95]. They asked for efficient algorithms for actually finding embeddings of low distortion and noticed that for some such problems, it is natural to use semi-definite programming. They applied this geometric connection to get old and new results for algorithmic problems on graphs (in particular, the sparsest cut problem we will soon discuss). Another motivation they discuss (which quickly developed into a major topic in approximation algorithms) is that some computations (e.g., finding nearest neighbors) are more efficient in some spaces than others, and so efficient, low-distortion embedding may provide useful reductions from a harder to an easier space. They describe such an efficient algorithm implementing Bourgain's Theorem 13.6, and also prove that his bound is the best possible (the metric proving it is simply the distances between points in any constant-degree *expander* graph; see Section 8.7).

The next shift in the evolution of this field, and in the level of interactions between geometers and ToC researchers, came from trying to prove "hardness of approximation" results. One example is the Goemans-Linial conjecture [Goe97, Lin02], studying the sparsest cut problem, about the relation between L_1 and the "negative type" metric space L_2^2 (a general class of metrics that arises naturally in several contexts).

---

[5]This beautiful conjecture states that for every d-regular graph, there exist $\{-1, 1\}$ signs of the edges, which make all eigenvalues of the resulting signed adjacency matrix lie in the "Ramanujan interval" $[-2\sqrt{d-1}, 2\sqrt{d-1}]$.

[6]With respect to the spectral gap. This is one of a few important expansion parameters to optimize.

[7]This is yet another example of a structural result (on doubly stochastic matrices) whose proof was partly motivated by algorithmic ideas. The connection is the use of hyperbolic polynomials in optimization (more specifically, as barrier functions in interior point methods.)

Roughly, these are metrics on $\mathbb{R}^n$ in which Euclidean distances are squared. More precisely, a metric (X,d) is of negative type (namely, in L_2^2), if $(X, \sqrt{d})$, is isometric (has no distortion) to a subset of L_2.

Conjecture 13.7. L_2^2 can be embedded into L_1 with constant distortion.

This conjecture was proved false by Khot and Vishnoi [KV05], who proved the following.

Theorem 13.8 [KV05]. *For every n there are n-point subsets of L_2^2 for which every embedding to L_1 requires distortion $\Omega(\log\log n)^{1/6}$.*

Far more interesting than the result itself is its origin. Khot and Vishnoi were trying to prove that the (weighted) "sparsest cut" problem is hard to approximate. They managed to do so using a computational assumption, known as the *unique games* conjecture of Khot [Kho02] (see also [Kho10] and Section 4.3), via a so-called *PCP*-reduction (see Section 10.3). The elimination of this computational assumption is the magical part that demonstrates the power and versatility of reductions between computational problems. They apply their PCP reduction to a particular, carefully chosen unique games instance, which cannot be well approximated by a certain semi-definite program. The outcome was an instance of the sparsest cut problem, which the same reduction ensures is hard to approximate by a semi-definite program. As discussed above, that outcome instance could be understood as a metric space, and the hardness of approximation translates to the required distortion bound.

The exact distortion of embedding L_2^2 into L_1 has been determined precisely to be $\sqrt{\log n}$ (up to lower order factors) in two beautiful sequences of works developing new algorithmic and geometric tools. I mention only the final word for each, as these papers contain a detailed history. On the upper bound side, the efficient algorithm approximating nonuniform sparsest cut to a factor $\sqrt{\log n}\log\log n$, which yields the same distortion bound, was obtained by Arora, Lee, and Naor [ALN08] via the so-called measured descent method. A lower bound of $\sqrt{\log n}$ on the distortion was very recently proved by Naor and Young [NY17] using a new isoperimetric inequality on the Heisenberg group.

Another powerful connection between such questions and ToC is through (again) expander graphs. A basic example is that the graph metric of any constant-degree expander proves that Bourgain's embedding theorem above is optimal. Much more sophisticated examples arise from trying to understand (and perhaps disprove) the Novikov and the Baum-Connes conjectures (see [KY06]). This program relies on another, much weaker notion of *coarse* embedding.

Definition 13.9. (X,d) has a coarse embedding into (X',d') if there is a map $f : X \to X'$ and two increasing, unbounded real functions α, β such that for every two points $x, y \in X$,

$$\alpha(d(x,y)) \le d'(f(x), f(y)) \le \beta(d(x,y)).$$

Gromov [Gro03] was the first to prove that an infinite family of expanders cannot be coarsely embedded into a Hilbert space. This result was greatly generalized by Lafforgue [Laf08] and Mendel-Naor [MN14], who constructed graph metrics that cannot be coarsely embedded into any *uniformly convex* space. It is interesting that while Lafforgue's method is algebraic, the Mendel-Naor construction follows the combinatorial *zig-zag* construction of expanders [RVW02] from computational complexity.

Many other interaction projects regarding metric embeddings and distortion that we do not touch on include their use in numerous algorithmic and data structure problems like clustering, distance oracles, and the k-server problem, as well as the fundamental interplay between distortion and *dimension reduction*, relevant to both geometry and CS, where so many basic problems are open.

13.5   Group theory

Group theorists, much like number theorists, have been intrinsically interested in computational problems since the origin of the field. For example, the *word problem* (given a word in the generators of some group, does it evaluate to the trivial element?) is so fundamental to understanding any group one studies, that as soon as language was created to formally discuss the computational complexity of this problem, hosts of results followed, trying to pinpoint that complexity. These include decidability and undecidability

results once Turing set up ToC and provided the first undecidable problems; these were followed by $\mathcal{NP}$-completeness results and efficient algorithms once $\mathcal{P}$ and $\mathcal{NP}$ were introduced around 1970. Needless to say, these *algorithmic results* inform us of the *structural* complexity of the groups at hand. And the word problem is but the first example. Another demonstration is the beautiful interplay between algorithmic and structural advances over decades, on the *graph isomorphism problem*, recently leading to the breakthrough of Babai [Bab15]![8] A huge body of work is devoted to finding efficient algorithms for computing commutator subgroups, Sylow subgroups, centralizers, bases, representations, characters, and a host of other important substructures of a group from some natural description of it. Excellent textbooks include [HEO05, Ser03].

Here we focus on two related problems, the *generation* and *random generation* problems, and the new conceptual notions borrowed from computational complexity that are essential for studying them. Before defining them formally, let us consider an example. Assume I hand you 10 invertible matrices, say 100×100 in size, over the field of size 3. Can you tell me if they generate another such given matrix? Can you even produce convincing evidence of this before we both perish? How about generating a random matrix in the subgroup spanned by these generators? The problem, of course, is that this subgroup will have size far larger than the number of atoms in the known universe, so its elements cannot be listed, and typical words generating elements in the group may need to be prohibitively long. Indeed, even in the extremely special cases, for elements in $\mathbb{Z}_p^*$ (namely one, 1×1 matrix), the first question is related to the *discrete logarithm* problem, and for $\mathbb{Z}_{p \cdot q}^*$, it is related to the *integer factoring* problem, both currently requiring exponential time to solve (as a function of the description length).

Let us consider any finite group G and let $n \approx \log |G|$ be roughly the length of a description of an element of G. Assume we are given k elements in G, $S = \{s_1, s_2, \ldots, s_k\}$. It would be ideal if the procedures we describe would work in time polynomial in n and k (which prohibits enumerating the elements of G, whose size is exponential in n).

The *generation problem* asks whether a given element $g \in G$ is generated by S. How does one prove such a fact? A standard certificate for a positive answer is a *word* in the elements of S (and their inverses) that evaluates to g. However, even if G is cyclic, the shortest such word may be exponential in n. An alternative, computationally motivated description, is to give a *program* for g. Its definition shows that the term "program" suits it perfectly, as it has the same structure as typical computer programs, only that instead of applying some standard Boolean or arithmetic operations, we use the group operations of multiplication and inverse.

Definition 13.10. A *program* (over S) is a finite sequence of elements $g_1, g_2, \ldots, g_m$, where every element g_i is either in S, or is the inverse of a previous g_j, or is the product of previous g_j, g_ℓ. We say that it "computes g simply" if $g = g_m$.

In the cyclic case, programs afford exponential savings over words in description length, as a program allows us to write large powers by repeatedly squaring elements. What is remarkable is that such savings are possible for *every* group. This discovery of Babai and Szemerédi [BS84] says that every element of every group has an extremely succinct description in terms of any set of elements generating it.

Theorem 13.11 [BS84]. *For every group G, if a subset of elements S generates another element g, then there is a program of length at most $n^2 \approx (\log |G|)^2$ that computes g from S.*

It is interesting to note that the proof uses a structure that is very combinatorial and counterintuitive for group theorists: that of a *cube*, which we will see again later. For a sequence $(h_1, h_2, \ldots, h_t)$ of elements from G, the cube $C(h_1, h_2, \ldots, h_t)$ is the (multi)set of 2^t elements $\{h_1^{\epsilon_1}, h_2^{\epsilon_2}, \ldots, h_t^{\epsilon_t}\}$, with $\epsilon_i \in \{0, 1\}$. Another important feature of the proof is that it works in a very general setting of "black-box" groups—it never needs an explicit description of the host group, only the ability to multiply elements and take their inverses. This is a very important paradigm for arguing about groups and will be used again below.

How does one prove that an element g is *not* generated by S? It is possible that there is no short "classical" proof! This question motivated Babai to define Arthur-Merlin games—a new notion of probabilistic, interactive proofs (simultaneously with Goldwasser, Micali, and Rackoff [GMR89], who proposed a similar

---

[8]See also Helfgott's exposition of this result in [HBD17].

notion for cryptographic reasons). Babai showed how nonmembership can be certified in this new framework. The impact of the definition of interactive proofs on the theory of computation has been immense and is discussed in Section 10.1.

Returning to the generation problem, let us now consider the more challenging problem of *random generation*. Here we are given S and would like a randomized procedure that will quickly output an (almost) uniform distribution on the subgroup H of G generated by S. This problem, besides its natural appeal, is often faced by computational group theorists, being a subroutine in many group-theoretic algorithms. In practice, heuristics are used, like the famous "product replacement algorithm" and its variants, which often work well (see, e.g., the recent [BLG12] and references). Here, we discuss provable bounds.

It is clear that sufficiently long random words in the elements of S and its inverses will do the job, but just as with certificates, sufficiently long is often prohibitively long.[9] In a beautiful paper, Babai [Bab91] describes a certain process generating a random program that computes a nearly uniform element of H and runs in time $n^5 \approx (\log |G|)^5$ steps.[10] This algorithm works in the full generality of black-box groups. This paper was followed by even faster algorithms with simpler analysis by Cooperman and by Dixon [Coo02, Dix08], and the state-of-the-art algorithm is one whose number of steps is, remarkably, the same as the length of proofs of generation mentioned above—in other words, randomness achieves the efficiency of nondeterminism for this problem.

Theorem 13.12 [Bab91, Dix08, Coo02]. *For every group G, there is a* probabilistic *program of length* $\text{poly}(n) \approx \text{poly}(\log |G|)$ *that, given* any *generating set S for G, produces with high probability a (nearly) uniformly random element of G.*

13.6   Statistical physics

The field of statistical physics is huge, and we focus here mainly on connections of statistical mechanics with ToC. Numerous mathematical models exist of various physical and chemical systems, designed to understand basic properties of different materials and the dynamics of basic processes. These include such familiar models as Ising, Potts, monomer-dimer, spin-glass, and Percolation. A typical example explaining the connection of such mathematical models to physics and chemistry, and the basic problems studied, is the seminal paper of Heilmann and Lieb [HL72].

Many of the problems studied can be viewed in the following general setting. We have a huge (exponential) space of objects called Ω (these objects may be viewed as the different configurations of a system). Each object is assigned a nonnegative weight (which may be viewed as the "energy" of that state). Scaling these weights gives rise to a probability distribution (often called the "Gibbs distribution") on Ω, and to study its properties (phase transitions, critical temperatures, free energy, etc.), one attempts to generate samples from this distribution (if the description of a state takes n bits, then listing all probabilities in question is exponentially prohibitive).

As Ω may be highly unstructured, the most common approach to this sampling problem is known as the Monte Carlo Markov Chain (or MCMC) method. The idea is to build a graph on the objects of Ω, with a pair of objects connected by an edge if they are similar in some sense (e.g., sequences that differ only in a few coordinates). Next, one starts from any object and performs a biased random walk on this graph for some time: The object reached is the sample produced. In many settings, it is not hard to set up the random walk (often named "Glauber dynamics" or the "Metropolis algorithm") so that the *limiting* distribution of the Markov chain is indeed the desired distribution. The main question in this approach is *when* to stop the walk and output a sample; *when* are we close enough to the limit? In other words, how long does it take the chain to converge to the limit[11]? In many cases, these decisions were taken on intuitive, heuristic grounds, without rigorous analysis of convergence time. The exceptions, where rigorous bounds were known, were typically structured, for example, when the Markov chain was a Cayley graph of certain groups (e.g., [Ald83, Dia88]).

---

[9]The reader familiar with Section 8.7 on expanders will realize that, if the Cayley graph of G with generators S is a sufficiently good expander, then short random words suffice.

[10]In a sense, the random straight-line program "adds generators" which shortcut long words and creates expansion.

[11]Yet again notice that this is essentially an issue of graph expansion (see Section 8.7) which we just encountered in the previous Section 13.5. Work described in this section has considerably enriched our understanding of expanders.

This state of affairs has changed considerably since the interaction in the past couple of decades with the theory of computation. Before describing it, let us see where computational problems even arise in this field. The two major sources are *optimization* and *counting*. That the setting above suits many instances of optimization problems is easy to see. Think of Ω as the set of solutions to a given optimization problem (e.g., the values of certain parameters designed to satisfy a set of constraints) and the weights as representing the quality of a solution (e.g., the number of constraints satisfied). So, picking at random from the associated distribution favors high-quality solutions. The counting connection is more subtle. Here Ω represents a set of combinatorial objects whose size one wants to count or approximate (e.g., the set of perfect matchings in a graph, or satisfying assignments to a set of constraints). A fundamental result of [JVV86] shows that for many such settings, sampling an object (approximately) at random allows a recursive procedure to yield an approximate the size of the set. Moreover, viewing the finite set as a fine discretization of a continuous object (e.g., lattice points in a convex set) allows one to compute volumes and more generally integrate functions over such domains.

Around 1990, rigorous techniques were introduced [Bro89, SJ89, Ald90, DFK91] to analyze the convergence rates of such general Markov chains arising from different approximation algorithms. They establish *conductance* bounds on the Markov chains, mainly via *canonical paths* or *coupling* arguments (a survey of this early work is [Jer96]). Collaborative work was soon able to formally justify the physical intuition behind some of the suggested heuristics for many models and moreover drew physicists to suggest such ingenious chains for optimization problems. The field drew in probabilists and geometers as well, and now is highly active and diverse. We mention two results to illustrate rigorous convergence bounds for important problems of this type.

Theorem 13.13 [JSV04]. *The permanent of any nonnegative $n \times n$ matrix can be approximated, to any multiplicative factor $(1 + \epsilon)$, in polynomial time in n/ϵ.*

The importance of this approximation algorithm stems from the seminal result of Valiant [Val79b] that the permanent[12] polynomial is *universal* for essentially all counting problems (in particular, those arising in the statistical physics models and optimization and counting problems mentioned above). So, unlike the determinant, computing it exactly is extremely difficult.

Theorem 13.14 [DFK91]. *The volume of any convex set in n dimensions can be approximated, to any multiplicative factor $(1 + \epsilon)$, in polynomial time in n/ϵ.*

The volume, besides its intrinsic interest, captures as well natural counting problems, for example, the number of extensions of a given partially ordered set. The analysis of this algorithm, as well as its many subsequent improvements, has led to purely structural results of independent interest in convex geometry as well as to generalizations like efficient sampling from any log-concave distribution (see the survey [Vem05]).

Another consequence of this collaboration was a deeper understanding of the relation between *spacial* properties (such as phase transitions and long-range correlations between distant sites in the Gibbs distribution) and *temporal* properties (such as speed of convergence of the sampling or approximately counting algorithms, like Glauber dynamics). This connection (surveyed, e.g., in [DSVW04]) has been established by physicists for spin systems since the 1970s. The breakthrough work of Weitz [Wei06] on the *hard core* model gave a *deterministic* algorithm that is efficient up to the phase transition, and this was complemented by a hardness result of Sly [Sly10] beyond the phase transition. These phase transitions of computational complexity, at the same point as the phase transitions of the Gibbs distribution, are striking, and the generality of this phenomenon is still being investigated.

More generally, the close similarity between statistical physics models and optimization problems, especially on random instances, is benefiting both sides. Let me mention a few exciting developments. This connection has unraveled the fine geometric structure of the space of solutions at the phase transition, pinpointing it, for example, for k-SAT [ACORT11]. Physics intuition based on such ideas as renormalization, annealing, and replica symmetry breaking has led to new algorithms for optimization problems, some of them now rigorously analyzed (e.g. [JS93]). Others, like one of the fastest (yet unproven) heuristics for such problems as Boolean satisfiability (which is $\mathcal{NP}$-complete in general), are based on the physics method of "survey propagation" [MPZ02, Het16]. Another connection is the *Lovász local lemma* (LLL), which enables one to establish the *existence* of rare "global" events. Efficient algorithmic versions of the LLL were initiated by Beck [Bec91], and starting with the work of Moser [Mos09] (and then [MT10]), have led to approximate counting and uniform sampling versions for rare events (see, e.g. [GJL16]). These new techniques for analyzing *directed, non-reversible* Markov chains are a powerful new tool for many more applications. A completely different *deterministic* algorithm of Moitra [Moi16] in the LLL regime promises many more applications; it works even when the solution space (and hence the natural Markov chain) is not connected. And back to MCMCs, we note the recent breakthrough work of [ALGV18] on approximating the partition function of the random cluster model and the number of bases in a matroid; in another wonderful demonstration of interconnectivity, it uses in particular results on random walks on *high-dimensional* expanders (mentioned at the end of our expanders section, Section 8.7).

Finally, we note that MCMC algorithms are among the oldest probabilistic algorithms, born and used way before computational complexity theory. However, with the study of randomness and its power in algorithms (to which Chapter 7 and Chapter 9 are devoted) our understanding of such algorithms grew considerably. A particularly interesting and surprising consequence of how few random bits actually are required by MCMC algorithms is given in [NZ96].

13.7   Analysis and probability

This section gives a taste of a growing number of families of inequalities—large deviation inequalities, isoperimetric inequalities, and the like—that have been generalized beyond their classical origins due to a variety of motivations in the theory of computing and discrete mathematics. Further, the applications sometimes call for *stability* versions of these inequalities, namely, an understanding of the structures that make an inequality nearly sharp. These motivations also pushed for generalizations of classical results and many new

---

[12]This notorious sibling of the determinant, in which no signs appear, was defined and discussed in Chapter 12.

ones. Most of the material here, and much more on the motivations and developments in this exciting area of the analysis of Boolean functions, can be found in the book by O'Donnell [O'D14].

The following story can be told from several angles. One is the *noise sensitivity* of functions. Let us restrict our attention to the Boolean cube endowed with the uniform probability measure, but many of the questions and results extend to arbitrary product probability spaces. Let $f : \{-1,1\}^n \to \mathbb{R}$, which we assume is balanced, namely, $E[f] = 0$. When the image of f is $\{-1,1\}$, we can think of f as a voting scheme, translating the binary votes of n individuals into a binary outcome. One natural desire from such a voting scheme may be *noise stability*—that typically very similar inputs (vote vectors) will yield the same outcome. While natural in this social science setting, such questions also arise in statistical physics settings, where natural functions, such as bond percolation, turn out to be extremely sensitive to noise [BKS99]. Let us formally define noise stability.

Definition 13.15. Let $\rho \in [0,1]$ be a correlation parameter. We say two vectors $x, y \in \{-1,1\}^n$ are ρ-correlated if they are distributed as follows. The vector x is drawn uniformly at random, and y is obtained from x by flipping each bit x_i independently with probability $(1-\rho)/2$. Note that for every i, the correlation $E[x_i y_i] = \rho$. The *noise sensitivity* of f at ρ, $S_\rho(f)$, is simply defined as the correlation of the outputs, $E[f(x)f(y)]$.

It is not hard to see that the function maximizing noise stability is any *dictatorship* function (e.g., $f(x) = x_1$) for which $S_\rho(f) = \rho$. But another natural social scientific concern is the *influence* of players in voting schemes [BOL85], which prohibits such solutions (in democratic environments). Influence of a voter is the probability with which it can change the outcome given that all other votes are uniformly random (so, in a dictatorship, it is 1 for the dictator and 0 for all others). A fair voting scheme should have no voter with high influence. As we define influence for real-valued functions, we will use the (conditional) *variance* to measure a player's potential effect given all other (random) votes.

Definition 13.16. A function $f : \{-1,1\}^n \to \mathbb{R}$ has influence τ if for every i, $\mathrm{Var}[x_i | x_{-i}] \leq \tau$ for all i (where x_{-i} denotes the vector x without the ith coordinate).

For example, the Majority function has influence $O(1/\sqrt{n})$. The question of how small the influence of a balanced function can be is extremely interesting and leads to a highly relevant inequality for our story (both in content and technique). As it turns out, ultimate fairness (influence $1/n$ per player) is impossible: [KKL88] shows that every function has a player with nonproportional influence, at least $\Omega(\log n/n)$. At any rate, one can ask which of the functions with *small* influence is most stable, and it is natural to guess that majority should be the best.[13]

The conjecture that this is the case, called the *majority is stablest* conjecture, arose from a completely different and surprising angle—the field of optimization, specifically "hardness of approximation." A remarkable paper [KKMO07] has shown that it implies[14] the optimality of a certain natural algorithm for approximating the maximum cut of a graph (the partition of vertices so as to maximize the number of edges between them—a basic optimization problem whose exact complexity is $\mathcal{NP}$-complete). This connection is highly nontrivial, but by now we have many examples showing how the analysis of certain (semidefinite programming-based) approximation algorithms for a variety of optimization problems raise many new isoperimetric questions, enriching this field.

The majority is stablest conjecture was proved in a strong form by Mossel, O'Donnell, and Oleszkiewicz in [MOO10] shortly after it was posed. Here is a formal statement (which actually works for bounded functions).

Theorem 13.17 [MOO10]. *For every (positive correlation parameter) $\rho \geq 0$ and $\epsilon > 0$, there exists (an influence bound) $\tau = \tau(\rho, \epsilon)$ such that for every n and every $f : \{-1,1\}^n \to [-1,1]$ of influence at most τ, $S_\rho(f) \leq S_\rho(Majority_n) + \epsilon$.*

The proof reveals another angle to the story—large deviation inequalities and invariance principles. To see the connection, recall the Berry-Esseen theorem [Fel71], generalizing the standard central limit theorem

---

[13]This noise sensitivity tends, as n grows, to $S_\rho(Majority_n) = \frac{2}{\pi}\arcsin \rho$.

[14]Assuming another, complexity-theoretic, conjecture called the "unique games" conjecture, discussed in Section 4.3.

to *weighted* sums of independent random signs. In this theorem, influences arise very naturally. Consider $\sum_{i=1}^{n} c_i x_i$. If we normalize the weights c_i to satisfy $\sum_i c_i^2 = 1$, then c_i is the influence of the ith voter, and $\tau = \max_i |c_i|$. The quality of this central limit theorem deteriorates linearly with the influence τ. Lindeberg's proof of the Berry-Esseen theorem uses an invariance principle, showing that for linear functions, the cumulative probability distribution $Pr[\sum_{i=1}^{n} c_i x_i \leq t]$ (for every t) is unchanged (up to τ), *regardless* of the distribution of the variables x_i, as long as they are independent and have expectation 0 and variance 1. Thus, in particular, they can be taken to be standard Gaussian, which trivializes the problem, as the weighted sum is a Gaussian as well!

To prove Theorem 13.17, one first observes that in the noise stability problem, the Gaussian case is simple. If the x_i, y_i are standard Gaussians with correlation ρ, the stability problem reduces to a classical result of Borell [Bor85]: that noise stability is maximized by any hyperplane through the origin. Note that here the rotational symmetry of multidimensional Gaussians, which also aids the proof, does not distinguish "dictator" functions from majority—both are such hyperplanes. Given this, an invariance principle whose quality depends on τ would do the job. The next step is show that it is sufficient to prove the invariance principle only for *low degree* multilinear polynomials (as the effect of noise decays with the degree). Finally, one needs to prove a nonlinear extension of Berry-Esseen for such polynomials, a form of which I state below. I note that the invariance principle was used to prove other conjectures in [MOO10], and since the publication of this paper, quite a number of further generalizations and applications have been found.

Theorem 13.18 [MOO10]. *Let x_i be any n independent random variables with mean 0, variance 1, and bounded 3rd moments. Let g_i be n independent standard Gaussians. Let Q be any degree-d multilinear n-variate polynomial of influence τ. Then for any t,*

$$|Pr[Q(x) \leq t] - Pr[Q(g) \leq t]| \leq O(d\tau^{1/d}).$$

To conclude this section, let me give one more, very different demonstration of the surprising questions (and answers) regarding noise stability and isoperimetry, arising from the very same computational considerations of optimization of hardness of approximation. Here is the question: *What is the smallest surface area of a (volume-1) body that tiles $\mathbb{R}^d$ periodically along the integer lattice $\mathbb{Z}^d$?* In other words, we seek a d-dimensional volume-1 subset $B \subseteq \mathbb{R}^d$ such that $B + \mathbb{Z}^d = \mathbb{R}^d$ and its boundary has minimal $(d-1)$-dimensional volume.[15] Let us denote this infimum by $s(d)$. You can pause here a bit and test your intuition: What do you expect the answer to be, asymptotically in d?

Such questions originate from the late nineteenth century study by Thomson (later Lord Kelvin) of *foams* in 3 dimensions [Tho87], further studied, generalized, and applied in mathematics, physics, chemistry, material science, and even architecture. However, for this very basic question, where periodicity is defined by the simplest integer lattice, it seems that, for large d, the trivial upper and lower bounds on $s(d)$ were not improved on for more then a century. The trivial upper bound on $s(d)$ is provided by the unit cube, which has surface area $2d$. The trivial lower bound on $s(d)$ comes from ignoring the tiling restriction and considering only the volume. In this case, the unit volume ball has the smallest surface area, $\sqrt{2\pi e d}$. Where in this quadratic range does $s(d)$ lie? In particular, can there be "spherical cubes" with $s(d) = O(\sqrt{d})$?

The last question became a central issue for complexity theorists when [FKO07] related it directly to the important unique games conjecture and the optimal inapproximability proofs of combinatorial problems (in particular, the maximum cut problem) discussed above. The nontrivial connection, which the paper elaborates and motivates, goes through attempts to find the tightest version of Raz's [Raz98a] celebrated parallel repetition theorem.[16] A limit on how "strong" a parallel repetition theorem can get was again provided by Raz [Raz11]. Extending his techniques to the geometric, continuous setting, [KORW08] resolved the question above, proving that spherical cubes do exist!

Theorem 13.19 [KORW08]. *For all d, $s(d) \leq \sqrt{4\pi d}$.*

---

[15] Note that the volume of B ensures that the interiors of $B + v$ and $B + u$ are disjoint for any two distinct integer vectors $u, v \in \mathbb{Z}^d$, so this gives a tiling.

[16] A fundamental information-theoretic inequality of central importance to the "amplification" of probabilistically checkable proofs (PCPs).

A simple proof and various extensions of this result were given subsequently in [AK09]. All known proofs are probabilistic. Giving an explicit construction that might better illustrate what a spherical cube (even with much worse but nontrivial surface area) looks like seems to be a challenging problem.

13.8   Lattice theory

Lattices in Euclidean space are among the most universal objects in mathematics, in that besides being natural (e.g., arising in crystalline structures) and worthy of study in their own right, they capture a variety of problems in different fields, such as number theory, analysis, approximation theory, Lie algebras, and convex geometry. Many of the basic results in lattice theory, as we shall see, are *existential* (i.e., supply no efficient means for obtaining the objects whose existence is proved), which in some cases has limited progress on these applications.

This section tells the story of one algorithm, of Lenstra, Lenstra, and Lovász [LLL82], often called the "LLL algorithm," and some of its implications for these classical applications as well as modern ones in cryptography, optimization, number theory, symbolic algebra, and more. But we had better define a lattice[17] first.

Let $B = \{b_1, b_2, \ldots, b_n\}$ be a basis of $\mathbb{R}^n$. Then the *lattice* $L(B)$ denotes the set (indeed, Abelian group) of all integer linear combinations of these vectors; $L(B) = \{\sum_i z_i b_i : z_i \in \mathbb{Z}\}$. B is also called a *basis* of the lattice. Naturally, a given lattice can have many different bases (e.g., the standard integer lattice in the plane, generated by $\{(0,1),(1,0)\}$, is equally well generated by $\{(999,1),(1000,1)\}$). A basic invariant associated with a lattice L is its determinant $d(L)$, which is the absolute value of $\det(B)$ for any basis B of L (this is also the volume of the fundamental parallelpiped of the lattice). For simplicity and without loss of generality, we will assume that B is normalized so that we only consider lattices L of $d(L) = 1$.

The most basic result about lattices, namely, that they must contain *short* vectors (in any norm) was proved by Minkowski (who initiated lattice theory, and with it, the geometry of numbers) [Min10].

Theorem 13.20 [Min10]. *Consider an arbitrary convex set K in $\mathbb{R}^n$ that is centrally symmetric[18] and has volume $> 2^n$. Then, every lattice L (of determinant 1) has a nonzero point in K.*

This innocent theorem, which has a simple but *existential* (pigeonhole) proof, turns out to have numerous fundamental applications in geometry, algebra, and number theory. Among famous examples, this theorem yields (with appropriate choice of norms and lattices) results like Dirichlet's Diophantine approximation theorem and Lagrange's four-squares theorem, and (with much more work) the finiteness of class numbers of number fields (see, e.g., [PZ89]).

From now on, we will focus on short vectors in the (most natural) Euclidean norm. A direct corollary of Minkowski's theorem when applying it to the cube $K = [-1, 1]^n$ yields the following.

Corollary 13.21. *Every lattice L of determinant 1 has a nonzero point of Euclidean norm at most $\sqrt{n}$.*

Digressing a bit, note that very recently, a century after Minkowski, a strong converse of the above corollary[19] conjectured by Dadush (see [DR16]) for *computational* motivation has been proved in [RSD16]. This converse has many structural consequences for the covering radius of lattices, arithmetic combinatorics, Brownian motion, and others. We will not elaborate here on this new interaction of computational complexity and optimization with lattice theory and convex geometry. The papers above beautifully motivate these connections and applications and narrate the history of ideas and technical work needed for this complex proof.

Returning to Minkowski's corollary for the Euclidean norm, the proof is still existential, and the obvious algorithm for finding such a short vector requires exponential time in n. The breakthrough paper [LLL82] describes the LLL algorithm, an efficient, polynomial-time algorithm, which approximates the length of the shortest vector in any n-dimensional lattice by a 2^n factor.

---

[17]Only full-rank lattices are defined here, which suffice for this exposition.

[18]Namely, $x \in K$ implies that also $-x \in K$. Such sets are precisely balls of arbitrary norms.

[19]Which has to be precisely formulated.

Theorem 13.22 [LLL82]. *There is a polynomial-time algorithm, which given any lattice L, produces a vector in L of Euclidean length at most a factor of 2^n longer than the shortest vector in L.*

This exponential bound may seem excessive at first, but the number and diversity of applications is staggering. First, in many problems, the dimension n is a small constant (so the actual input length arises from the bit-size of the given basis). This leads, for instance, to Lenstra's algorithm for (exactly solving) integer programming [Len83] in constant dimensions. It also leads to Odlyzko and Riele's refutation [OtR85] of Mertens' conjecture about cancellations in the Möbius function, and to the long list of number-theoretic examples in [Sim10]. But it turns out that even when n is arbitrarily large, many problems can be solved in poly(n)-time as well. Here is a list of examples of old and new problems representing this variety, some going back to the original paper [LLL82]. In all, it suffices that real number inputs are approximated to poly(n) digits in dimension n.

- **Diophantine approximation**. While the best possible approximation of one real number by rationals with bounded denominators is readily solved by its (efficiently computable) continued fraction expansion, no such procedure is known for *simultaneous* approximation. Formally, given a set of real numbers (say, $\{r_1, r_2, \ldots, r_n\}$), a bound Q, and $\epsilon > 0$, find integers $q \le Q$ and $p_1, \ldots, p_n$ such that all $|r_i - p_i/q| \le \epsilon$. Existentially (using Minkowski), the Dirichlet "box-principle" shows that $\epsilon < Q^{1/n}$, is possible. Using LLL, one efficiently obtains $\epsilon < 2^{n^2} Q^{1/n}$, which is meaningful for Q described by poly(n) many bits.

- **Minimal polynomials of algebraic numbers**. Here we are given a single real number r and a degree bound n, and are asked whether there is a polynomial $g(x)$ with integer coefficients of degree at most n, of which r is a root (and also to produce such a polynomial g if it exists). Indeed, this is a special case of the problem above with $r_i = r^i$. While the algorithm only outputs g for which $g(r) \approx 0$, it is often easy to check that it actually vanishes. Note that by varying n, we can find the minimal such polynomial.

- **Polynomial factorization over rationals**. Here the input is an integer polynomial h of degree n, and we want to factor it over $\mathbb{Q}$. The high-level idea is to first find an (approximate) root r of h (e.g., using Newton's method), feed it to the problem above, which will return a minimal g having r as a root, and thus divides h. We stress that this algorithm produces the exact factorization, not an approximate one!

- **Small integer relations between reals**. Given reals $r_1, r_2, \ldots, r_n$, and a bound Q, determine whether there exist integers $|z_i| < Q$ such that $\sum_i z_i r_i = 0$ (and if so, find these integers). As a famous example, LLL can find an integer relation among $\arctan(1) \approx 0.785398, \arctan(1/5) \approx 0.197395$, and $\arctan(1/239) \approx 0.004184$, yielding Machin's formula

$$\arctan(1) - 4\arctan(1/5) + \arctan(1/239) = 0.$$

- **Cryptanalysis**. Note that a very special case of the problem above (in which the coefficients z_i must be Boolean) is the "knapsack problem," a famous $\mathcal{NP}$-complete problem. The point here is that in the early days of cryptography, some systems were based on the assumed "average case" hardness of knapsack. Many such systems were broken by using LLL (e.g., [Lag84]). LLL was also used to break some versions of the RSA cryptosystem (with "small public exponents").

It is perhaps a fitting epilogue to the last item that lattices not only can destroy cryptosystems but also create them (we discuss cryptography in Chapter 18). The problem of efficiently approximating short vectors up to polynomial (as opposed to exponential, as LLL produces) factors is believed to be computationally hard. Here are some major consequences of this assumption. First, Ajtai showed in a remarkable paper [Ajt96] that such hardness is preserved on average, over a cleverly chosen distribution of random lattices. This led to a new public-key encryption scheme by Ajtai and Dwork [AD97] based on this hardness. Variants of this system, based on variants of this hardness assumption, are currently the only ones known that can

potentially withstand *quantum attacks*; we discuss quantum computing in Chapter 11, and in particular demonstrate how Shor's efficient quantum algorithms can efficiently factor integers and compute discrete logarithms [Sho94]. Recent progress in quantum computing technology has driven massive investment in making such lattice-based cryptosystems practical! In another breakthrough work of Gentry [Gen09b], this hardness assumption is used to devise *fully homomorphic* encryption, a scheme that allows one not only to encrypt data, but also to perform arbitrary computations directly with encrypted data. See more in the excellent surveys by Peikert [Pei16, Bra18].

13.9   Invariant theory

Invariant theory, born in an 1845 paper of Cayley [Cay45], is a major branch of algebra, with important natural connections not only to algebraic geometry and representation theory, but also to many other areas of mathematics. We will see some here, as well as some new connections with computational complexity, leading to new questions and results in this field. Note that computational efficiency has always been important in invariant theory, which is rife with ingenious algorithms (starting with Cayley's *Omega process*), as is evident from the books [CLO92, Stu08, DK15].

Invariants are familiar enough, from examples like the following:

- In high school physics, we learn that energy and momentum are preserved (namely, are *invariants*) in the dynamics of general physical systems.

- In chemical reactions, the number of atoms of each element is preserved as one mixture of molecules is transformed to yield another (e.g., as combining sodium hydroxide (NaOH) and hydrochloric acid (HCl) yields the common salt sodium chloride (NaCl) and water (H_2O)).

- In geometry, a classical puzzle asks: When can a plane polygon be "cut and pasted" along straight lines to another polygon? Here the obvious invariant, *area*, is the only one.[20] However, in generalizing this puzzle to 3-dimensional polyhedra, it turns out that besides the obvious invariant, *volume*, there is another invariant, discovered by Dehn.[21]

More generally, questions about the equivalence of two surfaces (e.g., knots) under homeomorphism, whether two groups are isomorphic, or whether two points are in the same orbit of a dynamical system, and so forth, all give rise to similar questions and treatment. A canonical way to give negative answers to such questions is through *invariants*, namely, quantities preserved under some action on an underlying space.

We will focus on invariants of *linear groups* acting on *vector spaces*. Let us go over some notation. Fix a field $\mathbb{F}$ (while problems are interesting in every field, results mostly work for infinite fields only, and sometimes just for characteristic zero or algebraically closed ones). Let G be a group, and V a representation of G, namely, an $\mathbb{F}$-vector space on which G acts: For every $g, h \in G$ and $v \in V$, we have $gv \in V$ and $g(hv) = (gh)v$.

The *orbit* under G of a vector (or point) $v \in V$, denoted Gv, is the set of all points that v can be moved to by this action, namely, $Gv = \{gv : g \in G\}$. Understanding the orbits of a group of actions is a central task of this field. A basic question capturing many of the examples above is: Given two points $u, v \in V$, do they lie in the same G-orbit, namely is $u \in Gv$? A related basic question, which is even more natural in algebraic geometry (when the field $\mathbb{F}$ is algebraically closed, and of characteristic zero), is whether the *closures*[22] of the two orbits intersect (namely, if some point in Gu can be approximated arbitrarily well by points in Gv). We will return to specific incarnations of these questions.

When G acts on V, it also acts on $\mathbb{F}[V]$, the polynomial functions on V, also called the *coordinate ring* of V. In our setting, V will have finite dimension (say, m), and so $\mathbb{F}[V]$ is simply $\mathbb{F}[x_1, x_2, \ldots, x_m] = \mathbb{F}[X]$, the polynomial ring over $\mathbb{F}$ in m variables. We will denote by gp the action of a group element $g \in G$ on a polynomial $p \in \mathbb{F}[V]$.

---

[20]And so, every two polygons of the same area can be cut to produce identical (multi)sets of triangles.

[21]So there are pairs of 3-dimensional polyhedra of the same volume, which cannot be cut to identical (multi)sets of tetrahedra.

[22]One can take closure in either the Euclidean or the Zariski topology (the equivalence in this setting has been proved by Mumford [Mum95]).

A polynomial $p(X) \in \mathbb{F}[X]$ is *invariant* if it is unchanged by this action; namely, for every $g \in G$, we have $gp = p$. All invariant polynomials clearly form a subring of $\mathbb{F}[X]$, denoted by $\mathbb{F}[X]^G$ and called the *ring of invariants* of this action. Understanding the invariants of group actions is the main subject of invariant theory. A fundamental result of Hilbert [Hil93] shows that in our linear setting,[23] *all* invariant rings will be *finitely generated* as an algebra.[24] Finding the "simplest" such generating set of invariants is our main concern here.

Two familiar examples of perfect solutions to this problem are the following.

- In the first, $G = S_m$, the symmetric group on m letters, is acting on the set of m formal variables X (and hence the vector space they generate) by permuting them. Then a set of generating invariants is simply the first m *elementary* symmetric polynomials in X.

- In the second, $G = SL_n(\mathbb{F})$, the simple linear group of matrices with determinant 1, is acting on the vector space $M_n(\mathbb{F})$ of $n \times n$ matrices (so $m = n^2$), simply by left matrix multiplication. In this case, all polynomial invariants are generated by a single polynomial, the determinant of this m-variable matrix X.

In these two cases, which really supply a complete understanding of the invariant ring $\mathbb{F}[X]^G$, the generating sets are *good* in several senses. There are *few* generating invariants, they all have *low* degree, and they are *easy* to compute[25]—all these quantities are bounded by a polynomial in m, the dimension of the vector space.[26] In such good cases, one has efficient algorithms for the basic problems regarding orbits of group actions. For example, a fundamental duality theorem of geometric invariant theory (see [MFK82], Theorem A.1.1), shows how generating sets of the invariant ring can be used for the orbit closure intersection problem.

Theorem 13.23 [MFK82]. *For an algebraically closed field $\mathbb{F}$ of characteristic 0, the following are equivalent for any two $u, v \in V$ and generating set P of the invariant ring $\mathbb{F}[X]^G$:*

- *The orbit closures of u and v intersect.*

- *For every polynomial $p \in P$, $p(v) = p(u)$.*

In the next subsections we will see how attempts to understand orbit closure intersection and related questions, even for very specific natural group actions, beautifully interact with computational complexity.

13.9.1   Geometric complexity theory

Here we briefly explain how, despite the fact that computation in general need not have any symmetries, the study of group actions as above can be very relevant to its core problems! This work has already generated many new questions and collaboration between the fields. First, some quick background on the main problem of arithmetic complexity theory (see Chapter 12 for definitions and more discussion). Valiant [Val79b] defined arithmetic analogs $\mathcal{VP}$ and $\mathcal{VNP}$ of the complexity classes $\mathcal{P}$ and $\mathcal{NP}$, respectively, and conjectured that these two arithmetic classes are different (see Conjecture 12.21). He further proved (via surprising completeness results) that to separate these classes, it is sufficient to prove that the permanent polynomial on $n \times n$ matrices does not project to the determinant polynomial on $m \times m$ matrices for any $m = \text{poly}(n)$. Note that this is a pure and concrete algebraic formulation of a central computational conjecture.

In a series of papers, Mulmuley and Sohoni introduced *geometric complexity theory* (GCT) to tackle this major open problem.[27] This program is surveyed by Mulmuley [Mul11, Mul12a], as well as in Landsberg's book [Lan17]. Very concisely, the GCT program starts off as follows. First, a simple padding of the $n \times n$

---

[23]The full generality under which this result holds are actions of *reductive* groups, which I do not define here, but includes all examples we discuss.

[24]This means that there is a finite set of polynomials $\{q_1, q_2, \ldots, q_t\}$ in $\mathbb{F}[X]^G$, and for every polynomial $p \in \mathbb{F}[X]^G$, there is a t-variate polynomial r over $\mathbb{F}$ so that $p = r(q_1, q_2, \ldots, q_t)$.

[25]For example, they have *small* arithmetic circuits or formulas.

[26]There are additional desirable structural qualities of generating sets that we will not discuss, such as completely understanding the algebraic relations between these polynomials (called *syzygies*).

[27]The origins of using invariant theory to argue computational difficulty via similar techniques go back to Strassen [Str87].

permanent polynomial makes it have degree m and act on the entries of an $m \times m$ matrix. Consider the linear group SL_{m^2} action on all entries of such $m \times m$ matrices. This action extends to polynomials in those variables, and so, in particular, to the two we care about: determinant and modified permanent. *The main connection is that the permanent projects to the determinant (in Valiant's sense) if and only if the orbit closures of these two polynomials intersect.* Establishing that they do not intersect (for $m = \mathrm{poly}(n)$) naturally leads to questions about finding representation-theoretic obstructions to such intersections (and hence, to the required computational lower bound). This is where things get very complicated, and describing them is beyond the scope of this survey. To date, the tools of algebraic geometry and representation theory have not been sufficient even to improve the quadratic bound on m of Theorem 12.25. Indeed, some recent developments show severe limitations to the original GCT approach (and perhaps guiding it to more fruitful directions); see [BIP16] and its historical account. Nevertheless, this line of attack (among others in computational complexity) has led to many new questions in computational commutative algebra and to growing collaborations between algebraists and complexity theorists—we will consider describe some of these now.

To do so, we focus on two natural actions of linear groups on *tuples* of matrices: simultaneous conjugation and the left-right action. Both are special cases of *quiver representations* (see [Gab72, DW06]).[28] For both group actions, we will discuss the classical questions and results on the rings of invariants, and recent advances motivated by computational considerations.

13.9.2   Simultaneous conjugation

Consider the following action of $SL_n(\mathbb{F})$ on d-tuples of $n \times n$ matrices. We have $m = dn^2$ variables arranged as d $n \times n$ matrices $X = (X_1, X_2, \ldots, X_d)$. The action of a matrix $Z \in SL_n(\mathbb{F})$ on this tuple is by simultaneous conjugation, by transforming it to the tuple $(Z^{-1}X_1Z, Z^{-1}X_2Z, \ldots, Z^{-1}X_dZ)$. Now, the general question for this action is: Which polynomials in the variables X are invariant under this action?

The work of Razmyslov, Procesi, Formanek, and Donkin [Raz74, Pro76, For84, Don92] provides a good set (in most aspects discussed above) of generating invariants over algebraically closed fields of characteristic zero. The generators are simply the traces of products of length at most n^2 of the given matrices[29] namely, the set

$$\{Tr(X_{i_1}X_{i_2}\cdots X_{i_t}) \,:\, t \leq n^2,\, i_j \in [d]\}.$$

These polynomials are explicit, have small degree, and are easily computable. The one shortcoming is the exponential size of this generating set. For example, using it to decide the intersection of orbit closures will only lead to an exponential time algorithm.

By Hilbert's existential Noether's normalization lemma [Hil93],[30] we know that the size of this set of generating invariants can, in principle, be reduced to $dn^2 + 1$. Indeed, when the group action is on a vector space of dimension m, taking $m+1$ "random" linear combinations of any finite generating set will result (with probability 1) in a small generating set. However, as we started with an exponential number of generators, this procedure is both inefficient and also not explicit (it is not clear how to make it deterministic). One can get an explicit generating set of minimal size deterministically using the Gröbner basis algorithm (see [MR11] for the best known complexity bounds), but this will take doubly exponential time in n.

The works above [Mul12b, FS13] reduce this complexity to polynomial time! This happened in two stages. First Mulmuley [Mul12b] gave a probabilistic polynomial-time algorithm by cleverly using the structure of the exponentially many invariants (using which, one can obtain sufficiently random linear combinations using only polynomially many random bits and in polynomial time). He then argues that using conditional derandomization results, of the nature discussed in Section 7.2, one can derive a deterministic polynomial-time

---

[28]I will not elaborate on the theory of quivers representation here, but only remark that reductions and completeness occur in this study as well. The left-right quiver is *complete* in a well-defined sense (see [DM15], Section 5). Informally, this means that understanding its (semi)-invariants implies the same understanding of the (semi)-invariants of all acyclic quivers.

[29]Convince yourself that such polynomials are indeed invariant.

[30]This is the same foundational paper that proved the *finite basis* and *Nullstellensatz* theorems. It is interesting that Hilbert's initial motivation to formulate and prove these cornerstones of commutative algebra was the search for invariants of linear actions.

algorithm under natural computational hardness assumptions. Shortly afterward, Forbes and Shpilka [FS13] showed how to de-randomize a variant of Mulmuley's algorithm without any unproven assumption, yielding an unconditional deterministic polynomial-time algorithm for the problem. Their algorithm uses the de-randomization methodology: Roughly speaking, they first notice that Mulmuley's probabilistic algorithm can be implemented by a very restricted computational model (a certain read-once branching program) and then use an efficient pseudo-random generator for this computational model. Here is one important algorithmic corollary (which can be extended to other quivers).

Theorem 13.24 [Mul12b,FS13]. *There is a deterministic polynomial-time algorithm for solving the following problem. Given two tuples of rational matrices $(A_1, A_2, \ldots, A_d)$, and $(B_1, B_2, \ldots, B_d)$, determine whether the closures of their orbits under simultaneous conjugation intersect.*

Interestingly if we only consider the orbits themselves (as opposed to their closure)—namely, ask whether there is $Z \in SL_n(\mathbb{F})$ such that for all $i \in [d]$ we have $Z^{-1}A_iZ = B_i$—this becomes the *module isomorphism* problem over $\mathbb{F}$. For this important problem, there is a deterministic algorithm (of a very different nature than above, using other algebraic tools) that can solve the problem over any field $\mathbb{F}$ using only a polynomial number of arithmetic operations over $\mathbb{F}$ [BL08].

13.9.3 Left-Right action

Consider now the following action of two copies, $SL_n(\mathbb{F}) \times SL_n(\mathbb{F})$, on d-tuples of $n \times n$ matrices. We still have $m = dn^2$ variables arranged as d $n \times n$ matrices $X = (X_1, X_2, \ldots, X_d)$. The action of a pair of matrices $Z, W \in SL_n(\mathbb{F})$ on this tuple is by left-right action, transforming it to the tuple $(Z^{-1}X_1W, Z^{-1}X_2W, \ldots, Z^{-1}X_dW)$. Which polynomials in the variables X are invariant under this action? Despite the superficial similarity to the simultaneous conjugation, the invariants here have a very different structure, and bounding their size requires different arguments.

The works [DW00,DZ01,SVdB01,ANS10] provide an infinite set of generating invariants. The generators (again, over algebraically closed fields) are determinants of linear forms of the d matrices, with *matrix* coefficients of arbitrary dimension:

$$\{\det(C_1 \otimes X_1 + C_2 \otimes X_2 + \cdots + C_d \otimes X_d) \,:\, C_i \in M_k(\mathbb{F}), k \in \mathbb{N}\}.$$

These generators, while concisely described, fall short on most of the goodness aspects mentioned above, and we now discuss improvements. First, by Hilbert's finite generation, we know in particular that some finite bound k exists on the dimension of the matrix coefficients C_i. A quadratic upper bound $k \leq n^2$ was obtained by Derksen and Makam [DM15] after a long sequence of improvements described there. Still, there is an exponential number[31] of possible matrix coefficients of this size that can be described explicitly, and allowing randomness, one can further reduce this number to a polynomial. Thus we have, for example, the following weaker analog to Theorem 13.24 regarding orbit closure intersection for this left-right action.

Theorem 13.25. *There is a probabilistic polynomial-time algorithm to solve the following problem. Given two tuples of rational matrices $(A_1, A_2, \ldots, A_d)$ and*
$(B_1, B_2, \ldots, B_d)$, determine whether the closures of their orbits under the left-right action intersect.

In the remainder of this section, we discuss an important special case of this problem, namely, when all $B_i = 0$, for which a *deterministic* polynomial-time algorithm was found. While this problem is in commutative algebra, this algorithm surprisingly has implications in analysis, non-commutative algebra, and beyond to computational complexity and quantum information theory. We will mention some of these, but let us start by defining the problem.

For an action of a linear group G on a vector space V, define the *nullcone* of the action to be the set of all points $v \in V$ such that the closure of the orbit Gv contains 0. The points in the nullcone are sometimes called *unstable*. The nullcone is of fundamental importance in invariant theory. Some examples of nullcones for actions we have discussed are the following. For the action of $SL_n(\mathbb{C})$ on $M_n(\mathbb{C})$ by left multiplication, it

---

[31]Well, a possibly infinite number, but it can be reduced to be exponential.

is the set of *singular* matrices. For the action of $SL_n(\mathbb{C})$ on $M_n(\mathbb{C})$ by conjugation, it is the set of *nilpotent* matrices. As you would guess (one direction is trivial), the nullcone is precisely the set of points that vanish under all invariant polynomials. Thus if we have a good generating set, we can use it to efficiently test membership in the nullcone. However, we are not able to do so for the left-right action. Despite this, a deterministic polynomial-time algorithm was obtained in [GGOW15] over the complex numbers, and then a very different algorithm was obtained in [IQS15] that works for all fields. These two algorithms have different natures and properties, and use in different ways the upper bounds on the dimension of matrix coefficients in the invariants.

Theorem 13.26 [GGOW15, IQS15]. *There is a deterministic polynomial-time algorithm, that on a given tuple of matrices $(A_1, A_2, \ldots, A_d)$ in $M_n(\mathbb{F})$ determines whether it is in the nullcone of the left-right action.*

We conclude with some of the diverse consequences of this algorithm. All the precise definitions of the notions below, as well as the proofs, interconnections, and the meandering story leading to these algorithms, can be found in [GGOW15, GGOW16].

Theorem 13.27 [GGOW15, GGOW16]. *There are deterministic polynomial-time algorithms to solve the following problems.*

- *The feasibility problem for Brascamp-Lieb inequalities, and more generally, computing the optimal constant for each.*

- *The word problem over the (non-commutative) free skew field.*

- *Computing the non-commutative rank of a symbolic matrix.*[32]

- *Approximating the commutative rank of a symbolic matrix to within a factor of 2.*[33]

- *Testing whether a completely positive quantum operator is rank-decreasing.*

---

[32]That is, a matrix whose entries are linear forms in a set of variables.

[33]Computing this rank exactly is the PIT problem discussed at the end of Section 12.4.

14 Space complexity: Modeling limited memory

Despite remarkable technological advances in miniaturizing computer memory (we are accustomed to carrying gigabytes of movies, pictures, and music in our pockets), space is a costly resource whose minimization is of importance in numerous applications, especially those dealing with "big data." One important message of this short chapter is that surprising things can be done with very little memory.

We start by describing the basic computational model of space-bounded algorithms, the main complexity classes studied, and some of the most basic results and open problems from the classical theory of space complexity. Then, to demonstrate the power of small space computation, we proceed in two frameworks. In the first, we discuss the more modern *streaming* model of space-bounded computation, in which a huge amount of data flies by, to be seen only once. Still, methods to "condense" it allow the computing of important statistics in small space. In the second, we describe two older results about the tiniest possible space and show counterintuitively, that a *fixed* amount of memory suffices to count to arbitrarily large numbers.

14.1 Basic space complexity

Space complexity is almost as well studied as time complexity (the main resource discussed in this book), and an elaborate complexity theory was developed to study the classes of problems solved by space-bounded algorithms and their relationship to time and other resource limitations. Indeed, some of these classes and connections have already been mentioned in this book, for example, the class $\mathcal{PSPACE}$ of strategic problems defined in Section 4.1, and the fundamental Theorem 10.3 that every problem in this class possesses an interactive proof. Most problems discussed in this book so far are in polynomial space, and a major question is which of them are solvable in much less space—sublinear, logarithmic in the input size, or even constant.

Modeling computation with so little memory, smaller than the input length, should be defined with care, so that space limitation is restricted precisely to the *working space* of the algorithm. The standard model simply distinguishes input access from working memory access. Thus, in a space-bounded algorithm (e.g., a Turing machine), the input resides on a *read-only* tape, which *cannot* be modified, and the space restriction applies only to a separate tape (or tapes) of working memory that have read/write access by the algorithm. Space $s(n)$ captures the class of problems solvable with such algorithms, say, Turing machines, that on input of length n use only $s(n)$ bits of their working memory. For problems requiring a long output (longer than the space bound), another separate *write-only* tape is provided.

By far the most important and well studied space-bounded complexity class is $\mathcal{L}$, consisting of problems solvable on $O(\log n)$ space. We list here a few basic problems in this class. The reader may want to find such small-space algorithms for them (most are quite simple).

- **Arithmetic problems.** Given two integers, compute their sum, product, and one modulo the other.

- **Comparison problems.** Compare two integers, sort a set of integers.

- **2-coloring.** Given a graph, determine whether it is bipartite.

- **Word problem in the free group.** [LZ77] Given a sequence from the alphabet $\{a, a^{-1}, b, b^{-1}\}$, determine whether their product is the identity.

As for time-bounded complexity, it is natural to extend the space-bounded model to allow probabilistic and nondeterministic computation.[1] Again, modeling should be done with care. To account correctly for space, the model should specify how the random/guess bits are accessed; it must ensure only one-time access to each such bit (which the algorithm may or may not want to explicitly store in working memory). The two standard (and equivalent) ways of thinking about this mechanism are (1) allowing the machine to have random/guess states that generate one coin-toss/guess-bit at a time, or (2) have all coin-tosses/guess-bits written on a separate tape that can be read from left to right (and so each bit is accessed once). This limited

---

[1]As well as alternating, interactive, quantum, and so forth, which we will not discuss.

access to randomness or nondeterminism will play a crucial role in the weakness of these models we discuss below, compared to their time-bounded analogs.

The probabilistic and nondeterministic analogs of $\mathcal{L}$ are denoted $\mathcal{BPL}$ (probabilistic log-space, analogous to $\mathcal{BPP}$), and $\mathcal{NL}$ (nondeterministic log-space, analogous to $\mathcal{NP}$), respectively. The basic observations that space upper bounds time, while time is never more than exponential in space, result in the following chain of inclusions between time and space classes:

$$\mathcal{L} \subseteq \mathcal{NL} \subseteq \mathcal{P} \subseteq \mathcal{NP} \subseteq \mathcal{PSPACE} \subseteq \mathcal{EXP}.$$

Just as more time buys more computational power (and, e.g., $\mathcal{P} \neq \mathcal{EXP}$), more space buys more computational power (and, e.g., $\mathcal{L} \neq \mathcal{PSPACE}$). These separations imply that some of the immediate inclusions above are strict, but we have no idea which.

To a large extent, space complexity is much better understood than time complexity. Let us mention a few such fundamental results and their intuitive meanings (without formally defining the related classes). Most of them are much easier to understand simply as low space algorithms for the following two variants, directed and undirected, of the *graph connectivity* problem.

The problems $DCONN$ and $UCONN$ are defined as follows. In both, the input is a graph G, together with two special nodes marked s and t, and the problem is to determine whether there is a path in G from s to t. The only difference is that in $DCONN$, the input graph is *directed*, and in $UCONN$ it is *undirected*. $DCONN$ plays for $\mathcal{NL}$ the role that SAT plays for $\mathcal{NP}$; it is a complete problem for this class. Of course, an appropriate notion of reduction has to be defined; here log-space reductions replace polytime reductions. The completeness of $DCONN$ follows simply from the fact that in the computation of a log-space machine, there are only polynomially many different *configurations*, and transitions between them on a given input are naturally described by a directed graph. An input is accepted iff an accepting configuration is reachable from the initial configuration. The simple fact that $DCONN$ has a polynomial-time algorithm explains the inclusion $\mathcal{NL} \subseteq \mathcal{P}$ mentioned above.

The undirected version $UCONN$ also played an important role as a complete problem for a class called $\mathcal{SL}$, by a similar argument to the one above. It was one of the important examples of problems in $\mathcal{BPL}$; but as we'll see below, this problem, and this class, are less special today due to Theorem 14.3 below.

We start with two upper bounds on the power of nondeterminism (or guessing) in the log-space regime. Recall that the analogous results for time complexity are not known, indeed, are not believed. The first, one of the oldest results in complexity theory, due to Savitch [Sav70], is the analog of $\mathcal{P} = \mathcal{NP}$. Here nondeterminism can be eliminated at a quadratic blow-up in space. This is achieved by showing that $DCONN$, the complete problem for $\mathcal{NL}$, can be solved deterministically using only $(\log n)^2$ space. Below we use the notation $\mathcal{L}^t$ to denote problems solvable by a deterministic algorithm using space $(\log n)^t$ for any rational t.

Theorem 14.1 [Sav70]. $\mathcal{NL} \subseteq \mathcal{L}^2$.

The next result, independently due to Immerman [Imm88] and Szelepcsényi [Sze88], is a strong analog of $\mathcal{NP} = \mathrm{co}\mathcal{NP}$ for space complexity. It says that existential and universal quantifiers can be exchanged at only *linear* cost in space. Equivalently, there is a deterministic log-space algorithm for the following task; it receives a directed graph as input, and produces another as output, such that one has an $s - t$ path if and only if the other one doesn't! A hint, for those who rise to the challenge of finding such an algorithm: It hinges on *counting* the number of paths from one node to another.

Theorem 14.2 [Imm88, Sze88]. $\mathcal{NL} = \mathrm{co}\mathcal{NL}$.

We next move to undirected connectivity. A breakthrough result of Reingold [Rei08] is a deterministic log-space algorithm for $UCONN$. This is one of the most sophisticated graph algorithms in existence, using expander graphs and pseudo-randomness in essential and surprising ways. More abstractly, the theorem says that *symmetric* nondeterminism adds no power for space-bounded algorithms.

Theorem 14.3 [Rei08]. $\mathcal{SL} = \mathcal{L}$.

We conclude with the (very low) cost of removing randomness from space-bounded algorithms, namely, derandomizing $\mathcal{BPL}$. It all started with the seminal construction of an *unconditional* pseudo-random generator

for probabilistic space-bounded computation of Nisan [Nis92]. Stressing again, unlike for time complexity (compare with Theorems 7.14, 7.15 in Section 7.3), there is no hardness assumption here. Indirectly, however, as should be expected by the connection between pseudo-randomness and hardness, at the heart of Nisan's construction is a *provable* lower bound (see more at the end of the next Section 14.2) To state the theorem, we note that pseudo-randomness for space is defined in the same way we defined it for time, but taking into account the 1-way access of randomness in space-bounded computation. We say that a distribution on n bits *fools* a log-space machine if 1-way access to its bits cannot be distinguished (say, with advantage $1/\text{poly}(n)$) from 1-way access to the uniform distribution on n bits.

Theorem 14.4 [Nis92]. *There is a log-space computable generator $G : \{0,1\}^m \to \{0,1\}^n$, with $m = O((\log n)^2)$, that fools log-space computations.*

Both theorems below follow by utilizing, in very different ways, Nisan's generator above. Both may be viewed as (incomparable) analogs to $\mathcal{BPP} = \mathcal{P}$. In the first result, the class $\mathcal{SC}$ contains all problems that can be solved by a deterministic algorithm of polynomial time and polylogarithmic space.

Theorem 14.5 [Nis94]. $\mathcal{BPL} \subseteq \mathcal{SC}$.

Theorem 14.6 [SZ99]. $\mathcal{BPL} \subseteq \mathcal{L}^{3/2}$.

From what we know, let's move to what we want to know. Perhaps the most outstanding question in basic space complexity is the exact power of randomness and nondeterminism in this model. The following conjectures reflect the popular belief that randomness adds no power at all, while nondeterminism does.

Conjecture 14.7. $\mathcal{BPL} = \mathcal{L}$.

An elegant "complete" variant of a graph-connectivity problem for which a log-space algorithm will prove this conjecture is given in [RTV06].

Conjecture 14.8. $\mathcal{NL} \neq \mathcal{L}$.

Algorithmically, this conjecture states that $DCONN$ does *not* have a deterministic log-space algorithm, unlike its undirected sibling $UCONN$.

As things stand, we know no natural function (say, in $\mathcal{NP}$) that requires more than logarithmic space to compute. Recall that we also know no natural function that requires more than linear time to compute. It was thus a bombshell when Fortnow [For00] gave a very simple proof that SAT, the most basic $\mathcal{NP}$-complete problem, is hard in the senses above for at least one of space or time. Indeed, the result is far stronger: Algorithms that use near-linear time must use almost linear space!

Theorem 14.9 [For00, FLvMV05]. *For every fixed $\epsilon > 0$, there exist $\delta > 0$, such that any algorithm solving SAT that uses $n^{1-\epsilon}$ space requires time at least $n^{1+\delta}$.*

Results of this nature are called *time-space trade-offs*, and are known for several other models besides Turing machines, both uniform and nonuniform. We will discuss another such trade-off in Section 15.2.2. Also see [BSSV03] and its historical overview.

14.2   Streaming and sketching

An exciting and challenging space-limited model, whose importance continually grows due to big data applications, is the *streaming* model (see the surveys [Mut05, Cor11, McG14]). Unlike classical space-bounded models that allow multiple access to the input, here the input "flies by," and what is not stored (in the limited space, which is far smaller than the input length) is gone forever. An example often cited to demonstrate situations motivating such a model is the experiments at the Large Hadron Collider (LHC) in Switzerland, recently famed for establishing the existence of the Higgs boson. Almost all of the enormous amount of data registered by the LHC detectors from the debris of the high energy collisions in this accelerator are discarded instantaneously for lack of space, and only a tiny amount is kept; the sophisticated algorithms used attempt to keep only information essential to the discovery of new phenomena. Needless to say, streams of vast amounts of data arise in numerous other experiments and observations in astronomy, biology, and

other sciences, as well as in the observation of Internet traffic, financial information, weather, and more. In almost all cases, only essential statistical or structural information about the data is required.

A down-to-earth example, among the first motivating ones for this field, is the following. Assume the input passing by is $x = (x_1, x_2, \ldots, x_n)$ with each x_i being an element in the range $[n] = \{1, 2, \ldots, n\}$. A basic question is how many *distinct* elements are in x. Of course, with linear $O(n)$ space, one can store x and answer this question, but what can be done if space is sublinear? It is pretty clear (but needs a proof) that it is impossible to solve this problem exactly and deterministically, and indeed, most algorithms in this field allow both approximation and randomization. For example, for the problem above, what do you think is the minimal amount of space needed to give, with probability at least 99%, an estimate within 1% of the correct number of distinct elements? The answer is surprisingly small: $O(\log n)$ bits suffice for such a high quality estimate with such high confidence. Such an algorithm was discovered by Alon, Matias, and Szegedy [AMS96], and the reader is invited to try to find a sublinear space algorithm.

Indeed, [AMS96] and subsequent papers studied such algorithms for computing other "frequency moments" of the distribution x. More precisely, the input x defines a histogram $h = (h_1, h_2, \ldots, h_n)$, where h_i counts the number of x_j having the value i. The number of distinct elements of x is simply the number of nonzero entries of h, or in other words, the "0-norm" of h, denoted $|h|_0$. Some of the most valuable statistical information about such numeric data are other norms (or moments) $|h|_p$ of h, or the entropy of this distribution, and so forth. As it happens, some can be estimated accurately in small space, and others cannot, where both upper and lower bounds on space often require quite sophisticated techniques and lead to interesting connections (e.g., to topics like stable distributions, metric embeddings, sparse recovery, and more). The most common method used in streaming algorithms is the maintenance of a *sketch* (or fingerprint), a small space data structure that captures a sufficient amount of information about the part of the input seen so far, which is easy to update upon the arrival of a new data item. Needless to say, finding an appropriate sketch for a given problem is highly nontrivial. Much more on streaming and sketching, for these and other types of problems (e.g., approximating parameters on graphs) can be found in the surveys mentioned above.

Let me give a (high level) demonstration of a sketch in action. This elegant choice, for approximating the L_2 norm of h above, was suggested by Indyk [Ind00]. Again, the reader might consider the challenge of doing so in very small space, a task that looks impossible initially, before reading on. Here is the algorithm. Before seeing the input, we pick once and for all a random sign vector in $v \in \{-1, 1\}^n$. The sketch will be a single integer z, initialized at 0. It will be updated with each arrival of an input x_i as follows: If $x_i = j$, we simply add v_j to z. It is easy to see by linearity that at the end of this algorithm, the value of z is simply the inner product of v and h. As v was a random sign vector, the expectation of z^2 is $|h|_2^2$, and moreover, it is highly concentrated around its mean. So, to get an accurate estimate with high probability, one applies the usual trick: Maintain a few independent vectors v and sketches z for each, and at the end of the algorithm, report their median value. Note that each of the z counters requires only $O(\log n)$ space. Are we done? Wait (I hear you say)—what about the random vectors v? Storing them requires linear space! Actually, it does not—Indyk shows in his paper how to use Nisan's space pseudo-random generator of Theorem 14.4 to generate and store good enough alternative vectors using only $O(\log n)^2$ space, giving yet another application of the de-randomization quest discussed in Chapter 7.

Even from this simple example, a few consequences about the streaming and sketching "way of thinking" are evident. One is that a sketch can provide a highly accurate estimate not only at the end of the input, but actually throughout its arrival, after every new symbol is added. Thus, the input length need not be fixed in advance and can be viewed as infinite. Another is that the input may be viewed differently than some raw data from a scientific observation. For example, it may be a sequence of updates to an object (e.g., a network in which certain links and nodes are lost or added over time), and a sketch may capture various connectivity properties of the network. Thus sketches give rise to new data structures for dynamic problems in which the input changes with time.

Let's conclude with a simple streaming task that *cannot* be solved in small space; indeed, it is the very problem on which Nisan's pseudo-random generator in Theorem 14.4 is based. Two random sequences, first x and then y, of length $10s$ each, fly by, and the task is computing their inner product modulo 2. Try proving the following statement: the probability of any space s algorithm, guessing their inner product,

cannot exceed the trivial $\frac{1}{2}$ by more than 2^{-s}. Such lower bounds often use *communication complexity*, the topic of Chapter 15, where we further discuss Nisan's generator.

14.3   Finite automata and counting

The most severe restriction on memory is to bound it by an absolute constant, independent of input size. This section illustrates that even such a limited model has surprising power, which in turn demonstrates an important point: Some lower bounds may be hard to prove for the simple reason that they are false. For example, the two algorithms below were discovered when attempting to prove they do not exist. We will also see that the power of this very weak computational model is far from completely understood; indeed some of the open questions regarding it are closely related to questions on standard space complexity classes discussed above.

The "impossible" task that demonstrates this power is the basic problem of *counting*. It is not hard to show that counting arbitrarily high is impossible with a fixed amount of memory. This intuition seems to extend to the finite-memory devices which can toss random coins, or obtain some input-independent "advice" (please check whether you believe these intuitive statements). But we will see that in both cases, a few bits of memory (say, 10) suffice to determine whether a binary sequence of *any* length has more 1's than 0's (with high probability, say $2/3$, in the probabilistic setting).

Let us describe the computational model a bit more conveniently. When the working space $s(n)$ of a Turing machine is of constant size (namely, independent of the input size n), it can be simply viewed as part of the finite control of the machine. In other words, the Turing machine becomes a *finite automaton*. Finite automata played an important role in the history of computer science. Formal definitions and central results as well as their role in the early development of computation theory, can be found in Sipser's excellent textbook [Sip97].

We start with a 2-way deterministic finite automaton (2DFA), which is precisely a Turing machine with constant space. The "2-way" stresses that the input tape can be scanned in both directions (like a Turing machine, but in contrast to other automata we shall soon meet). A finite automaton has a finite control (or program) that is captured by a finite number of states. The automaton is initially at some starting state, with its reading head on the (say) left input symbol. At each step, the cell contents and the current state determine the next state of the automaton (if it enters Accept or Reject state, then it halts), and if the head moves left or right (we assume that the left and right endpoints of the input are detectable). Summarizing, a 2DFA is simply a Turing machine without the ability to write.

A 2DFA cannot count! Consider the Majority function, which for a binary sequence computes if it has more 1's than 0's. Try proving that a 2DFA with n states cannot compute Majority even for sequences of length $2n$. So, we will enhance our 2DFA with extra capabilities, as discussed above. Thus, let us thus try to add power or extra capabilities to our finite automata: nondeterminism, alternation, randomness, and advice, and discover the origins of this activity, so fundamental to our field (as we have seen throughout this book).

But before starting, let us discuss 1DFA, the older sibling of 2DFA. This model was defined by Kleene [Kle51] to understand the "nerve nets" of McCulloch and Pitts [MP43], the first mathematical model of neurons and the brain. Kleene proved that 1DFAs compute precisely *regular* languages: sets of sequences possessing strong periodicity structure.[2] This characterization easily implies that Majority is not a regular language and cannot be computed by 1DFAs.

The first to systematically explore the relative power of different finite automata models with different capabilities were Rabin and Scott, in their seminal paper [RS59], which set an example for many later studies of other computational models. In particular, Rabin and Scott [RS59] defined 2DFAs, and one of their major results was that for deterministic computation, the two models are equal in power.[3] This proves

---

[2]A *regular language* S (over an alphabet Σ) is either a finite set of sequences over Σ, or can be obtained from previously defined languages as the union of two, $S = S_1 \cup S_1$, the concatenation of two, $S = S_1 S_2$ (concatenating any sequence of S_1 to one in S_2), or the *Kleene star* of one, $S = S_1^*$ (namely, a concatenation of any finite number of sequences of S_1).

[3]This is far from obvious, as a 2DFA can access the input bits arbitrarily many times, whereas a 1DFA sees each input bit for one step, and then it is gone. Try proving it!

that 2DFAs, like 1DFAs, compute exactly the regular languages. And in particular, 2DFAs cannot compute Majority either.

In the same paper, Rabin and Scott suggested adding *nondeterminism* to the model (this inspired the later addition of nondeterminism to Turing machines, e.g., when extending $\mathcal{P}$ to $\mathcal{NP}$) by allowing several possible transitions from each given state and cell contents. They also proved that the resulting model, called "2NFA," cannot compute more sets than its deterministic sibling.[4] The model was further extended to allow *alternation* (in the same way the single existential quantifier in $\mathcal{NP}$ is extended to alternating existential and universal quantifiers in the definition of the polynomial time hierarchy $\mathcal{PH}$) in [LLS84]. They proved, however, that even this model, called 2AFA, still computes no more sets than 2DFA, namely, only regular languages. Summarizing, starting with the weakest deterministic one-way automaton, 1DFA, and upgrading it with 2-way input access, nondeterminism, and even alternation, does not increase the computational power.

So, we turn to adding *randomness* to the models, creating, respectively, 1PFA and 2-PFA for one-way and two-way probabilistic finite automata. As in Turing machines, these models are allowed to toss perfect random coins and use them in computation (namely, take a random transition between states), and are required to compute the correct answer with high probability (say, 2/3) on every input. Can this model do more than all the others above? The results just mentioned seem to hint that if space is constant, then it is hard to utilize extra capabilities, like nondeterminism and alternation. Furthermore, in the 1-way model, Rabin [Rab63] proved that 1PFA cannot do more than 1DFA, so adding randomness does not help in that setting either: 1PFA can only compute regular languages. It thus came as a surprise when Freivalds [Fre81] proved that 2-way probabilistic finite automata *are* stronger! In particular, 2PFAs can count arbitrarily high.

Theorem 14.10 [Fre81]. *There is a 10-state 2PFA that computes Majority with probability $\geq 2/3$ on every input. Moreover, for every $\epsilon > 0$, there is an integer $c = c(\epsilon)$ and a c-state 2PFA that computes Majority with probability $\geq 1 - \epsilon$ for every input.*

We will explain the simple idea behind this algorithm at the end of the section, leaving you time and space to try and figure it out yourself.

We proceed to add a different feature to 2-way automata: *nonuniformity*. Nonuniformity is discussed in Chapter 5. There, the nonuniform model of Boolean circuits was shown to be equivalent to the uniform Turing machines when equipped with (input-independent) *advice*. More specifically, there and here, we allow the machine, when given an input of length n, to have (read-only) access to an external (advice) tape of some fixed polynomial length in n. Such a nonuniform machine computes a function if for every n, there exists a binary (advice) sequence α_n, which if it resides on the advice tape, causes the machine to give a correct answer on every length n input (no requirement is made if the advice tape contains any other string). Note that the advice sequences α_n can of course depend on the function computed, but being so short, cannot contain the answer to every possible input of length n.

So, what good can an advice sequence of (say) length n^{10} do you, if you have only 10 (or even a million) bits of memory and are required to compute Majority on an arbitrary sequence of n bits? The community working on this problem was unanimous that such short advice is useless, and that in fact, an exponential lower bound on the length of the advice string should not be difficult to prove (indeed, such a lower bound was proved if the machine has only *1 bit of memory*). It thus came as a shock when Barrington (who was trying to prove such a lower bound) announced that this task is possible [Bar86]. Indeed, Majority can be computed in this model with only *3 bits of memory*! And many other functions as well!

Theorem 14.11 [Bar86]. *There is a 5-state nonuniform 2DFA[5] that computes the Majority function. In-*

---

[4]They actually proved it for 1NFA, simulating it by a 1DFA, but the same proof works for 2NFA. Note that this simulation of nondeterministic automata by deterministic ones incurs exponential blow-up in the number of states. This is known to be tight for 1NFA, but it is completely open for 2NFA. Indeed, proving a super-polynomial lower bound on the number of states in such a simulation will prove Conjecture 14.8. See [Pig13] for a survey of this approach.

[5]In accordance with our notation in Definition 5.2 for (nonuniform) circuits, we might denote this nonuniform automata by 2DFA/poly, to stress that the advice length is polynomial in the input length. The more common name for this model is *constant-width, polynomial size branching programs*.

deed, the same holds for every function computed by a polynomial-size Boolean formula.

I suspect that this proof (which is short and sweet) will be harder to rediscover than the previous one, so let me give you a few hints. A significant hint, and the main insight of the proof, is using in an essential way a nonsolvable group (the number 5 of states is related to the alternating group on 5 letters being the smallest nonsolvable permutation group). Another hint is that the automaton constructed is *reversible* (not only is the next configuration uniquely determined from the previous one, given the input symbol read, as in any deterministic machine, but also the previous is determined from the next). Finally, don't try to prove the first statement about Majority, but rather the second, as it allows induction on the structure of the formula computed. The advice used by the 2PFA should be interpreted as a sequence of which input bits are to be read at what step; the length of the advice sequence is quadratic in the formula size.

Let me add two more comments about this remarkable result. First, it inspired an analogous result for arithmetic computation, showing how to evaluate an arithmetic formula with a constant number of registers (indeed, three suffice) [BOC92]. Second, the reversibility aspect of formula evaluation was critical to many uses of Theorem 14.11 in cryptography, starting with [GMW87, Kil88].

Now let's return to probabilistic automata, and conclude with the idea behind Freivalds' Theorem 14.10. You have (say) 10 bits of memory and a coin to toss. Presented with a binary sequence (say, for simplicity, of odd length), scan it from left to right, and perform the following "tournament" between the 0's and 1's you observe. Toss a coin for every bit you see, and separately record whether it comes up Heads for all 0's (call this event W_0), or if it comes up Heads for all 1's (call this event W_1). This requires only one bit of memory each. If neither or both events happened, repeat this tournament again (this decision requires a couple more bits of memory). If exactly one happened, declare that one as the loser (namely, if W_0 happened, output that there is a majority of 1's, and vice versa). The analysis is easy: the probability of the event for the minority bit is at least twice as high as the one for the majority bit, and so this algorithm will be correct with probability $2/3$. To boost the success probability, and prove the second part of the theorem, simply toss t coins per input bit. This will require t extra states but will boost the ratio between the probability of the two events from 2 to 2^t.

The observant reader will have noticed a few unsatisfactory properties of this algorithm. For one, it may never halt. This is necessary; indeed, a 2PFA that always halts computes only regular languages, as proved in the original paper [Fre81]. Of course, with probability 1 it does halt, but in expected exponential time. This too is necessary; Dwork and Stockmeyer [DS90] proved that if a 2PFA halts in expected polynomial time, then it computes only regular languages.

An intriguing question that remains open is the power of this probabilistic polynomial time, constant space model in the interactive proof setting, namely, when the 2-way automata are allowed both randomness and nondeterministic transitions. Does this model only compute regular languages? This question was raised in [DS92], and further progress on it was made in [CHPW98].

15   Communication complexity: Modeling information bottlenecks

This field of communication complexity studies the communication costs of computing *discrete* functions whose input is split between two parties. It was born in 1979 with a paper by Yao [Yao79], following similar work of Abelson on *continuous* functions. The field grew rapidly, both mathematically and applications-wise, and the comprehensive book of Kushilevitz and Nisan [KN97] covers the first two decades of activity. Today, two decades after that book was published, there is a dire need for a new book summarizing the amazing work that has been done since, with some very recent breakthroughs, including solutions to very old problems and exciting new directions.

The purpose of this chapter is not to summarize all that, but rather to focus on a single feature of the communication complexity model: its versatility. When I was a student, I witnessed a conversation between another student and Andy Yao regarding this model. Frustrated that Yao's original paper gives almost no motivation, the student asked him why computer scientists should study such a simple, stylized model, which is purely information theoretic and in particular ignores all computational aspects. Yao's answer was simple: because it is *basic*. To me, starting a career in research, this answer was a lesson for life. The ensuing discoveries of many diverse *computational* settings to which this model provides crucial insight reaffirms this lesson again and again. In this chapter, we'll see these applications to VLSI design, auction theory, circuit complexity, linear programming, pseudo-randomness, data structures, and more. Needless to say, communication *is* an important computational resource in distributed systems—but in some of these applications, we will see that through simple or subtle reductions, it informs us about other computational resources like time, space, size, randomness, queries, and chip area. Not surprisingly, in nearly all cases, the proof proceeds by a reduction, which has the following nature: A model of computation that uses too little of a given resource (or combination of resources) to compute a given function is shown to exhibit a communication bottleneck, which allows converting this computation into a cheap communication protocol for a related communication task.

After all that, I can't resist describing some new exciting and fundamental work that actually further pursues the connections of communication complexity with information and coding theory: *compression* and *error correction* of communication protocols.

15.1   Basic definitions and results

I give here a very short overview of some basic aspects of communication complexity, aiming at simplicity rather than generality, and at results we shall use in the applications discussed below. As mentioned, a comprehensive text for its day is [KN97], and some more recent excellent surveys on different aspects of this growing field include [Lov14,She14,Rou16]. I present here only Yao's basic model. In the bullets on different applications below, we will see how each plays a different variation on this simple basic theme.

A communication problem is simply a 2-argument function, $f : X \times Y \to Z$, where X, Y, and Z are finite sets. An input $x \in X$ is given to Alice, and an input $y \in Y$ is given to Bob. Together, they should both compute $f(x, y)$ by exchanging bits of information in turn, according to a previously agreed-on protocol. There is no restriction on their computational power; the only measure we care to minimize is communication cost. We formalize these notions below, first for deterministic protocols and then for probabilistic ones. In the spirit of computational complexity theory, other "modes" beyond deterministic and probabilistic were borrowed from Turing machines and adapted to the communication model, including nondeterministic, alternating, quantum, Arthur-Merlin, and others. Indeed, Babai, Frankl, and Simon [BFS86] initiated the definitions and systematic study of the related complexity classes. We will not discuss these extensions here.

A *deterministic protocol* specifies for each of the two players the bit to send next, as a function of their input and history of the communication so far, as well as the output at termination. It can be naturally described as a binary tree, where internal vertices are labeled by Boolean functions on X or Y (depending whose turn it is to speak at this node), and the leaves are labeled by the output value (in Z). The communication complexity (or cost) of a protocol is simply its depth (which counts the maximum number

of bits exchanged on any input). It computes a function f if for every input pair (x, y) the path followed by the players' communication on this input arrives at a leaf labeled $f(x, y)$. The *deterministic communication complexity* of a communication function f is the cost of the cheapest protocol computing it. We denote this quantity by $D(f)$.

A convenient way to view a communication function f is as a matrix M_f whose rows are labeled by elements of X, columns labeled by elements of Y, and the (x, y) entry is $f(x, y)$. It is convenient to understand the effect of communication protocols on this matrix. A basic insight is that when the first bit is sent (say, from Alice to Bob), its value partitions the rows X (Alice's possible inputs) in two parts, creating a submatrix on which they proceed. In it, Bob's next bit to Alice partitions Y, and so forth. As this process continues, we see that every c-bit protocol induces a partition of the matrix M_f into (at most) 2^c "combinatorial rectangles," where a rectangle is simply a Cartesian product of a subset $X' \subseteq X$ and $Y' \subseteq Y$. Moreover, if the protocol computes f, then all rectangles in this partition are *monochromatic*, namely, labeled by a unique element of Z. This insight is the source of all deterministic lower bounds. An especially useful observation, due to Mehlhorn and Schmidt [MS82], is that if we view Z as a subset of a field K, then $D(f) \geq \log \mathrm{rk}_K(M_f)$, where rk_K denotes the rank function in this field.[1] Another useful observation we shall soon use is that when the matrix is triangular with a nonzero diagonal, its rank is full.

Let us look at a few examples of natural functions, some of which will show up below, and gain some intuition about the model. In all cases, we take $X = Y = \{0, 1\}^n$, and $Z = \{0, 1\}$. Communication complexity is measured as a function of the input size n. Note from the outset that in this model, $n + 1$ is an upper bound on the communication complexity of every communication function (as one of them can send its input to the other, which will compute the answer and send it back).

1. **Equality:** $EQ(x, y) = 1$ iff $x = y$.

2. **Greater-or-equal:** $GE(x, y) = 1$ iff $x \geq y$.

3. **Disjointness:**[2] $DISJ(x, y) = 0$ iff for some i, $x_i = y_i = 1$.

It is easy to see that for all three functions, their matrices are triangular[3] with 1's on the diagonal, and so by the above rank lower bound, they all have essentially maximal deterministic communication complexity.

Fact 15.1.

1. $D(EQ) \geq n$.

2. $D(GE) \geq n$.

3. $D(DISJ) \geq n$.

We now allow the players to toss coins, which adds considerable power (sometimes). A *probabilistic protocol* is simply a distribution over deterministic protocols.[4] Its cost is the maximum depth of any tree in the support of this distribution. Such a protocol computes a function f with error ϵ if for every input pair (x, y), the probability that a random protocol from this distribution reaches a leaf labeled $f(x, y)$ is at least $1 - \epsilon$. As before, the *probabilistic communication complexity* of a communication function f is the cost of the cheapest probabilistic protocol computing it in this sense. We denote this quantity $R_\epsilon(f)$. In most cases, we pick $\epsilon = \frac{1}{3}$, and let $R(f) = R_\epsilon(f)$ (error reduction can be achieved by independent repetition, as in probabilistic algorithms). Lower bounds on probabilistic communication complexity, which are extremely useful in many applications, typically demand much more sophisticated techniques; some of these and their relative power are discussed in [KMSY14, LS09].

---

[1]How tight this lower bound is in general is the subject of the notorious *log-rank conjecture*; see [Lov14] for history and state-of-the-art.

[2]Here x, y are viewed as characteristic vectors of subsets of $[n]$.

[3]For disjointness, it is the bottom *right* of the matrix that is all zeros, when rows and columns are sorted lexicographically.

[4]This is sometimes called the *shared randomness* model (which samples for both players the protocol to use). A *private randomness* model (in which each player tosses its own coins) is also used, but the two differ very slightly in complexity, within additive $O(\log n)$ [New91].

The probabilistic communication complexity of the three functions above, which were equivalent in the deterministic model, turn out to be very different from each other.

Theorem 15.2.

1. $R(EQ) = O(1)$.

2. $R(GE) = \Theta(\log n)$ *(e.g., see [Vio15])*.

3. $R(DISJ) = \Theta(n)$ *[KS92, Raz92, BJKS04]*.

The upper bounds in these theorems are very simple and highlight the importance of *hashing* in randomized protocols. For example, for the first result on EQ, assume Alice has x, Bob has y, and they share a common random sequence r, all of length n. Then $\langle x, r \rangle$ and $\langle y, r \rangle$ provide 1-bit hash values of x, y respectively, which are always equal if $x = y$, and are different with probability $\frac{1}{2}$ if $x \neq y$. For the second result on GE, one can use the same hashing idea on segments of the two inputs to discover, via binary search, the most significant bit on which x and y differ, thus determining which is bigger. Of course, for the third, the upper bound is trivial.

The lower bounds merit a longer discussion. Probabilistic lower bounds are typically much harder to prove and are much more useful in applications. The first step in almost all such proofs (as well as for many other types of probabilistic algorithms) follows the so-called Yao's minimax principle [Yao77], namely, considering the following dual question. Rather than focusing on the best probabilistic protocol for a worst-case input, we focus on the best deterministic protocol for an average-case input. This allows proving a lower bound on deterministic protocols, when the input is chosen at random according to some distribution.

More precisely, let μ be any distribution on $X \times Y$. The *distributional* communication complexity of a function f under distribution μ with error ϵ, denoted $C_{\mu,\epsilon}(f)$, is the number of bits in a protocol that computes $f(x, y)$ correctly with probability $1 - \epsilon$, when (x, y) are drawn according to μ. It is easy to see that for every distribution μ, we have a lower bound $R_\epsilon(f) \geq C_{\mu,\epsilon}(f)$. Moreover, as Yao [Yao77] points out, equality is satisfied for some distribution μ^*, which achieves $R_\epsilon(f) = C_{\mu^*,\epsilon}(f)$; this setup is simply a special case of von Neumann's minimax theorem for zero-sum games [vN28]. Thus, any distribution will give a lower bound, and there is no loss of generality in this approach, as some distribution will give an optimal lower bound. Again we suppress ϵ when setting it equal to $\frac{1}{3}$; namely, we denote $C_\mu(f) = C_{\mu,1/3}(f)$.

Choosing a good distribution is not an obvious matter. Let us consider the lower bounds above. For the second one on GE, picking $\mu = \mu_X \times \mu_Y$ to be any *product* distribution (in which x and y are chosen independently) will lead to[5] $C_{\mu_X \times \mu_Y}(GE) = O(1)$. However, for a choice of μ that correlates[6] the two inputs (x, y), one gets a tight lower bound $C_\mu(GE) = \Omega(\log n)$ [Vio15]. For the third lower bound on $DISJ$, taking again a product distribution will not suffice; [BFS86] proves that $C_{\mu_X \times \mu_Y}(DISJ) = O(\sqrt{n} \log n)$. A correlated choice μ leads to the optimal result above: $C_\mu(DISJ) = \Omega(n)$. The three (difficult!) proofs in [KS92, Raz92, BJKS04] are quite different in language and style. But they all follow a similar intuition, and they all have an important feature in common that we shall later use. It is that the hard distribution they pick for $DISJ$ is supported on pairs of sets that are either disjoint or have a single intersection! We state this important distributional lower bound explicitly.

Theorem 15.3 [KS92, Raz92, BJKS04]. *There is a distribution[7] μ on $\{0,1\}^n \times \{0,1\}^n$, supported on pairs of sequences x, y with at most one coordinate that is 1 in both, such that $C_\mu(DISJ) = \Omega(n)$.*

While I motivated distributional communication complexity as a tool to prove probabilistic lower bounds, it is of course interesting in its own right, as there are certainly natural situations when an input distribution is given or known. At any rate, proving distributional lower bounds, even though we can now assume the

---

[5]Please check this at least for the uniform distribution.

[6]In a simple-to-guess way: Pick uniformly at random an index $i \in [n]$, and sequences $z \in \{0,1\}^i, w, w' \in \{0,1\}^{n-i}$, and set $x = zw$, $y = zw'$.

[7]We describe the distribution μ chosen in [BJKS04]. Pick pairs of bits (x_i, y_i) uniformly and independently at random from the set $\{(0,0), (0,1), (1,0)\}$. With probability $\frac{1}{2}$, give the players the resulting x, y, respectively. With probability $\frac{1}{2}$, give the players these inputs, but flip the ith coordinate in both to 1, for a randomly chosen $i \in [n]$.

protocol is deterministic, is typically difficult! There are many techniques, such as the *discrepancy bounds*, *corruption bounds*, and smooth and relative variants of these. We will not discuss them here, and address one lower bound idea, through *direct sum*, when we discuss information complexity in Section 15.3.1.

The task of proving lower bounds on communication models, and more generally understanding their relative power, has seen tremendous growth recently, due to a technique called *lifting*. It literally allows lifting lower bounds for much weaker models (like decision tree depth or polynomial degree) into communication complexity lower bounds. See e.g. [GPW15, GPW17] and their references for more on this technique, and its consequences.

15.2   Applications

We now describe some of the many models for which information is a bottleneck, and the variants of the basic communication complexity model needed to prove these limitations. Note that some of these and many others appear, in much more detail, in the beautifully written monograph of Roughgarden [Rou16].

15.2.1   VLSI time-area trade-offs

VLSI stands for very large scale integration. This semiconductor-based technology, developed in the 1970s, still dominates the fabrication of integrated circuits, like the ones in the microprocessor chips operating your phones and laptops. Today we can pack billions of transistors on one such microprocessor the size of a postage stamp, which was far from being the case in the early days. But both then and now, finding ways of utilizing the area of a chip optimally is crucial.[8] The earliest application of communication complexity (really, before the model was fully formalized) was showing that there is an inherent trade-off between the area of a chip computing a function and the time it takes to compute it. This important connection is the PhD work of Thompson [Tho79, Tho80], which was later extended in many other works.

Let us specify the computational model. To eliminate technology from the discussion, we assume that the computing elements of the chip reside on a grid of unit length, and that sending a bit between neighboring computing elements takes unit time. Each computing element can store a constant number of bits, and in one unit of time can compute an arbitrary function of its memory and send a bit to any of its neighbors. The initial placement of input bits is arbitrary, and so is the placement of the output bit(s).

The communication complexity measure of a function g we will need is denoted $C(g)$ (sometimes called the *arbitrary partition* communication complexity of g), and is defined as follows. Assume $g : \{0,1\}^{2n} \to \{0,1\}$ is a function on $2n$ bits. Every subset $S \subseteq [2n]$ of size n naturally defines a communication function $f_S : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$, obtained by giving the inputs in S to Alice and those outside S to Bob, and asking them to compute g. Now $C(g)$ is defined as the minimum of the communication complexity $D(f_S)$ over all such partitions S.

Theorem 15.4 [Tho79]. *Every function g computed by a chip of area A in time T satisfies $AT^2 \geq (C(g))^2$.*

The proof is simple. A geometric argument shows that in any chip of area A, and any subset of its computing elements of size $2n$, there must be a way to cut the chip into two parts along grid lines, so that the length of the cut is at most $\sqrt{A}$ and each part contains n elements of the given subset. Clearly, if the input bits initially reside in this subset, $C(g)$ bits must flow across the cut, and so $\sqrt{A}T \geq C(g)$.

Thompson has used this argument to prove quadratic AT^2 lower bounds for the Fourier transform, which is a multi-output function. However, it is easy to see that there are simple functions g with $C(g) = \Omega(n)$ (which is the largest possible), yielding $AT^2 \geq n^2$ lower bounds for such functions.[9] Put differently, a chip with too low AT^2 product gives a communication protocol of too low a cost!

---

[8]This is a general comment about technology, which explains the (unreasonably short) shelf life of any hi-tech product: As its speed and storage increase, so does the desire to apply it to larger problems (larger input data, finer resolution, etc.).

[9]Obtaining such functions g (that are hard for any partition) from any communication function f (that is hard for a fixed partition) is done simply by encoding the partition S into g's input. This only doubles the input length.

15.2.2   Time-space trade-offs

As you may recall, we still have no nontrivial lower bounds on either the time or the space required to compute explicit functions. In a similar vein to the previous Section 15.2.1, we try proving at least that both resources cannot be very small at the same time. We now prove a result of this nature, for a model called the *oblivious branching program (OBP)*. Intuitively, an OBP accesses the bits of the input (possibly multiple times) in a *fixed* order, independent of their value. More precisely, an OBP of space S and time T computes a function $h : \{0,1\}^n \to \{0,1\}$ if there is a sequence $\sigma \in [n]^t$ and a read-only space s Turing machine, which on input $x \in \{0,1\}^n$ reads the input bits $x_{\sigma(1)}, x_{\sigma(2)}, \ldots, x_{\sigma(t)}$ and outputs the value $h(x)$.

The minimal resources for an OBP are constant space and linear time. The following result of Alon and Maass [AM88] proves that they cannot be simultaneously achieved for the Majority function, and in particular, for constant space, time $\Omega(n \log n)$ is necessary.

Theorem 15.5 [AM88]. *For every n, any space S, and time T, OBP computing the Majority function on n bits satisfies $ST \geq \Omega(n \log n)$.*

We note that the surprising Theorem 14.11, which we describe in Section 14.3,[10] implies that Majority does have an OBP with constant space and polynomial time, indeed, smaller than n^5. There is still a large gap between the upper and lower bounds, and getting tighter bounds on the complexity of this basic function would be very interesting.

For a natural function that is a bit more complex than Majority, the same paper [AM88] proves a much better time-space trade-off. For simplicity, we define this function over a 3-letter alphabet, although it can be made Boolean. Let $Palindrome : \{0, 1, *\}^n \to \{0, 1\}$ be the function, which is 1 iff the $0, 1$ pattern of the input (ignoring the *'s) is a palindrome.

Theorem 15.6 [AM88]. *For every n, any OBP computing the Palindrome function on inputs of length n in space S and time T must satisfy $T = \Omega(n \log(n/S))$. In particular, if $S = o(n)$ then T is super-linear .*

The proof of both theorems uses the same "Ramsey-theoretic" lemma (interesting in its own right and with other applications), which enables embedding a hard communication complexity problem into the computation. Intuitively, every short enough sequence on $[n]$ must have two large subsets whose occurrences alternate few times. Here the length of the sequence will be the time T, which is assumed wlog to be a multiple of n.

Lemma 15.7 [AM88]. *For every n, k, for every sequence $\sigma \in [n]^{nk}$ there exist two disjoint subsets $A, B \subseteq [n]$ such that $|A| = |B| = n' = n/2^{8k}$ and σ contains no k-long subsequence alternating between elements of A and B.*

Let us see how a space S OBP (for one function) can give rise to a low communication protocol (for a related function) and use communication lower bounds to derive OBP lower bounds. Observe that if the inputs bits in A are given to Alice, and those in B to Bob, any fixing of the remaining bits to constants defines a function of communication complexity at most Sk (each alternation can be simulated by sending S bits from one player to the other). Theorems 15.5 and 15.6 are now easy to deduce. For the Palindrome function, simply set the inputs outside A, B to *'s, resulting in the equality function on n' bits (which has communication complexity n'). For the Majority function, set the bits outside A, B to have an equal number of 0's and 1's, resulting in the Greater-or-Equal functions on $\log n'$ bits, which has communication complexity $\log n'$.

Remember that time-space trade-offs for *general* Turing machines (as opposed to the weaker, *oblivious* ones considered here), were later proven using completely different methods—see Theorem 14.9.

15.2.3   Formula lower bounds

We now show how to use communication complexity arguments to prove lower bounds on the size of Boolean formulas. This connection was discovered in the paper of Karchmer and Wigderson [KW90].

---

[10]Where OBPs are viewed as 2-way finite automata with advice.

The basic definitions and background regarding Boolean formulas are given in Section 5.2.2, but it suffices here to recall that we deal with formulas over the standard logical connectives $\{\wedge, \vee, \neg\}$ (e.g., $(x \vee \bar{y}) \wedge z$) and that a *monotone* formula can't use negation (it can use only $\{\wedge, \vee\}$). Also, a basic result of Spira [Spi71] shows that formulas, and monotone formulas, can always be "balanced," making their depth logarithmic in their size.

The following two (essentially tight) lower bounds follow via the connection above, for the connectivity and perfect matching functions on graphs (which of course are monotone functions). In both, n denotes the number of vertices of the input graph.

Theorem 15.8 [KW90]. *Every monotone formula for testing whether a graph is connected requires size* $n^{\Omega(\log n)}$.

Theorem 15.9 [RW92]. *Every monotone formula for testing whether a graph has a perfect matching requires size* $2^{\Omega(n)}$.

To explain the connection to communication complexity, we need to extend the notion of communication problems from computing functions to computing relations.

For finite sets X, Y, Z, a relation $F \subseteq X \times Y \times Z$ defines a communication problem in which Alice gets as input some $x \in X$, Bob gets $y \in Y$, and they should compute some $z \in Z$ satisfying $(x, y, z) \in F$. We only consider relations for which there is at least one legal answer z for every input pair (x, y).

The deterministic communication complexity $D(F)$ and the probabilistic communication complexity $R(F)$ of a relation F are defined in the same way they are defined for functions.

The key idea in connecting formula complexity to communication complexity is the following association of a communication relation F_g (sometimes called a "KW-game") to every Boolean function g. Moreover, if g is monotone, one can assign to it another (harder) communication relation denoted by F_g^{mon}. We now define both.

Suppose that $g : \{0, 1\}^m \to \{0, 1\}$. For both relations, F_g and F_g^{mon}, set $X = g^{-1}(0)$, $Y = g^{-1}(1)$, and $Z = [m]$. In other words, in both, Alice receives a 0-input of g, Bob receives a 1-input of g, and they should compute an input coordinate (that we will denote as i rather than z) of some variable. The only difference is which indices are legal answers:

- $(x, y, i) \in F_g$ iff $x_i \neq y_i$.

- $(x, y, i) \in F_g^{mon}$ iff $x_i < y_i$.

Note that in both relations, there is always at least one legal answer for every input pair (x, y). As x and y always have different g values, they must be different and so must have at least one coordinate on which their inputs disagree. Moreover, if g is monotone, it is easy to see that at least in one coordinate the difference will be in the order dictated.

Let us denote by $d(g)$ the *depth* of the shallowest formula for a function g. Similarly, we denote by $d^{mon}(g)$ the depth of the shallowest monotone formula for a monotone function g. The main connection states simply the depth of g and communication complexity of F_g are always equal, and the same holds in the monotone case.

Theorem 15.10 [KW90].

- *For every function g, $d(g) = D(F_g)$.*

- *For every monotone function g, $d^{mon}(g) = D(F_g^{mon})$.*

This theorem has a simple inductive proof and is left to the reader. In brief, a formula for g and a communication protocol for F_g are two views of the same object. Both are described by binary trees, and the key conceptual difference (which in some cases makes proving lower bounds for protocols easier) is that in formulas computation is perceived as starting at the leaves and propagating to the root; in communication protocols, the computation is perceived as starting at the root and ending at a leaf.

With this connection, and recalling a simple fact[11] that a formula's depth is (up to a constant) the logarithm of the size, Theorems 15.8 and 15.9 follow from Theorems 15.11 and 15.12, where communication lower bounds on the associated relations are proved. Let Conn and PM denote, respectively, the connectivity and perfect matching functions on graphs. Note that for graphs of n vertices, the input size of these functions is $m = n^2$. While lower bounds on the deterministic communication complexity suffice, we actually know that they hold even for probabilistic communication, a fact that will come in handy soon.

Theorem 15.11 [KW90, RW89]. $R(F_{Conn}^{mon}) = \Omega((\log n)^2)$.

Theorem 15.12 [RW92]. $R(F_{PM}^{mon}) = \Omega(n)$.

While I will not prove these theorems, I wish to address one important point. Given the simplicity of the equivalence between formula depth and communication complexity, one may ask what advantage the communication complexity viewpoint brings. The answer in both cases is different (below) but what is common to both, as well as to other proofs, is simply that we have an arsenal of tools and results in communication complexity whose descriptive language fits perfectly two communicating players but is completely obscure if translated to formulas. These tools specialize in two different ways for the proofs of the theorems above.

For the first lower bound on graph connectivity, essential use is made of the basic fact that in the communication complexity framework, we have *two* inputs, while the formula has only *one*. The proof combines top-down induction with random restriction arguments to prove that the players cannot even solve the problem on "large enough" subsets of their respective inputs.

The second lower bound on perfect matching is proved by a direct reduction to the set-disjointness function $DISJ$ discussed in Section 15.1. As the reduction is probabilistic, one needs a lower bound for the randomized communication complexity of this problem, obtained in Theorem 15.3, which was luckily discovered shortly before this work. It is hard to imagine using such a result in the context of formulas, where this disjointness lower bound is meaningless.

Many other lower bounds were proved on these and other monotone computation models using the communication complexity method, especially with the recent advance of the so-called *lifting* method. The reader can learn more from [RPRC16, PR17].

So far we have seen only monotone lower bounds. How about nonmonotone ones? Observing the striking similarity between the monotone and general KW-relation of a given monotone function, it seems likely that a minor modification to a lower bound on a monotone relation can perhaps be "fixed" to provide a nonmonotone lower bound. To see that the two are quite different, here is one major feature in which they differ. For every function g on m bits, the randomized communication complexity of F_g is at most $2 \log m$ (this follows from the first item in Theorem 15.2), whereas for some g, like the perfect matching function above, we saw a $\sqrt{m}$ lower bound. Thus in particular, to prove nonmonotone lower bounds, one cannot use distributional arguments (in which inputs are chosen at random), nor can one use probabilistic reductions to other problems.

It might be hard, but here is a concrete challenge. Prove that if Alice has an n-bit prime, and Bob has an n-bit composite, then they have no deterministic $O(\log n)$ bit communication protocol to find a coordinate in which their inputs differ. Clearly, proving this implies that testing primality has no polynomial-size Boolean formula.

15.2.4 Proof complexity

Although this application is mostly self-contained, the reader might want to review Chapter 6 on proof complexity, especially Section 6.3.2 on the cutting planes proof systems and Section 6.4 on the feasible interpolation method.

The aim of proof complexity is to show that natural propositional tautologies require long proofs in natural proof systems. For simplicity, we define everything here *semantically*, even though syntax is crucial in proof systems. Consider Boolean functions on n variables. We say that two Boolean functions f, g imply

---

[11]Via the balancing of binary trees.

a third one h (denoted by $f, g \vdash h$) if every x satisfying both f, g also satisfies h; specifically, $f(x) = 1$ and $g(x) = 1$ implies $h(x) = 1$.

Now let F be any family of Boolean functions on n variables that contains the constant 0 function (F roughly corresponds to the proof system at hand). A subset $A \subseteq F$ is a *contradiction* if there is no input x satisfying all functions in A. A tree-like *refutation* for A is a binary tree whose nodes are labeled by functions from F satisfying the following conditions:

- Every leaf is labeled by a function from A.

- The root is labeled by the constant function 0.

- If h labels a node, and f, g are the labels of its children, then $f, g \vdash h$.

Note that any such refutation is indeed a logical proof that A is a contradiction. We seek natural contradictions A (typically, having poly(n) "simple" functions) for which the *size* (e.g., number of leaves) of every tree-like refutation is large (hopefully, super-polynomial or exponential in n). Let's describe a general way to prove *depth* lower bounds on refutations, which was proposed by Pudlák and Buss [PB94]. Its value is that for some families F, such as the one we will focus on soon, a lower bound of d on the depth will imply a lower bound of $\exp(d)$ on the size—which is the result we are seeking.

An F-query is simply an evaluation of any function $f \in F$ on any input. Consider a player who is given an input x and wishes to find a function $a \in A$ such that $a(x) = 0$ (there must be one, if A is a contradiction). Let $Q_F(A)$ be the smallest number of queries needed to solve this search problem on the worst case input x. It is obvious that a lower bound on $Q_F(A)$ gives a lower bound on the depth of any tree-like refutation of A.[12]

We are now ready to explain the connection to communication complexity. The above solitary game describes what is commonly called a *decision tree*, albeit here the set of allowed queries is nonstandard,[13] and it computes a relation and not a function. Now assume that the bits of input x are actually divided between two players, Alice and Bob (e.g., $x = y, z$, with Alice receiving y and Bob z). Further assume that under this input partition, *all* functions $f \in F$ have low communication complexity (e.g., $D(f) \leq c$ or $R(f) \leq c$). Then a depth d refutation of A now translates to a $2dc$ (deterministic or probabilistic, respectively) communication protocol for the original search problem. In other words, communication complexity lower bounds imply proof complexity lower bounds, which will be especially useful if the upper bound c is small.

This idea was used by Impagliazzo, Pitassi, and Urquhart [IPU94] to prove a lower bound on tree-like cutting-planes refutations as follows. First, the cutting-planes proof is captured by setting the family F to be all linear inequalities with integer coefficients on the variables x_i of the input x. More formally, for every linear inequality of the form $\sum_i s_i x_i \leq t$ (with t, s_i integers),[14] $f \in F$ assigns 1 to every x satisfying this inequality and 0 otherwise. It should be clear[15] that the communication complexity of any such $f \in F$ reduces to that of the function GE, namely, testing which of two $n \log n$-bit integers is larger. We have seen in the second item of Theorem 15.2 that $R(GE) = O(\log n)$.

Next, [IPU94] proves that for this cutting plane F, any tree-like proof can be balanced; thus, depth lower bounds imply size lower bounds. Finally, Impagliazzo et al. need a contradiction, which requires high communication complexity. Luckily, the proof of Theorem 15.12 already implicitly contains such a contradiction. The monotone KW-game above for the perfect matching problem immediately suggests such a "perfect matching" contradiction, which is easily expressed by linear inequalities. I will not write this contradiction down explicitly here. Together with the $\Omega(n)$ lower bound on the probabilistic communication complexity in Theorem 15.12 (which was derived from Theorem 15.3), Impagliazzo et al. derive the required exponential lower bound.

---

[12] Simply put, if we had a refutation of depth d, our player could start at the root (where the 0 function falsifies its input x) and proceed down to a leaf, each time querying the functions labeling the children of the current node, and proceeding to any that falsifies x. This will require $2d$ queries.

[13] The most well-studied case, called a Boolean decision tree, is when queries are simply the coordinate functions x_i.

[14] Which, without loss of generality, are no more than $n \log n$ bits in length; verify this fact!

[15] Alice and Bob can each separately compute the partial sum on their input bits.

Theorem 15.13 [IPU94]. *The "Perfect Matching" contradiction on n-vertex graphs requires cutting-planes tree-like contradictions of size* $\exp(n/\log n)$.

15.2.5   Extension complexity

One of the most ingenious and unexpected applications of communication complexity comes from the paper of Yannakakis [Yan91]. In this paper, Yannakakis connects it with convex geometry, and shows how *extension complexity* of every high-dimensional polytope (which I will define soon) is captured *exactly* by the nondeterministic communication complexity of a related communication problem. This understanding has since played an important role in polyhedral combinatorics and optimization.

But before I tell you that story, I have to tell you this story. Every year, the ToC community is exposed to many new papers claiming to solve the $\mathcal{P}$ vs. $\mathcal{NP}$ problem (in much the same way the math community is exposed to claimed solutions of the Riemann hypothesis). Luckily, almost all such papers display blatant errors or misunderstandings and so can be dismissed without much time investment. Occasionally, however, such papers are not easy to dismiss: They seem to contain real ideas, are rigorously presented, and (who knows?) may contain a proof. Dealing with these presents a nontrivial challenge to the community, as the claims are of the utmost interest. This is the story of one such attempt to prove $\mathcal{P} = \mathcal{NP}$, by Swart [Swa86].

The idea was simple and powerful. The *traveling salesman problem (TSP)* is $\mathcal{NP}$-complete. It can be written as a linear program, a problem that, a few years earlier, was found to be in $\mathcal{P}$ (as discussed at the end of Section 3.2). The main issue is that when written as a natural linear program in the *original variables* (the edges of a given graph), it has exponentially many linear constraints. What Swart suggested is adding *auxiliary variables*, and he presented a new linear program, with only polynomially many constraints, with these variables added. He claimed that the new linear program also captures TSP, and hence $\mathcal{P} = \mathcal{NP}$. It was not easy, but some dedicated researchers found some errors in Swart's program. Swart suggested a fixed program, but then more bugs were found in this one as well, and so on. When should the community give up, when the solution to its most fundamental problem is possibly within reach?

Yannakakis' paper [Yan91] set out to prove that Swart's approach has to fail *in principle*. To discuss this paper, we must first formulate this approach mathematically. We need a few preliminaries. A *polytope* $P \subseteq \mathbb{R}^n$ is the convex hull of a finite set of points $V \subseteq \mathbb{R}^n$. Another convenient description of a polytope P is as the intersection of a finite number of halfspaces F, called *facets*. We assume that both V and F are *minimal*: removing an element of either yields a smaller polytope than P. Let $|F| = f$ and $|V| = v$. The description of P via its facets gives f inequalities, concisely described by the *linear program*, $P = \{x : Ax \leq b\}$, where $x \in \mathbb{R}^n$, $b \in \mathbb{R}^f$, and A is an $f \times n$ real matrix. The basic decision problem, testing whether P is empty (as well as optimizing over P, which we will not discuss) can be solved in polynomial time in the dimensions of A.

The following example helps illustrate the complexity issues. Consider the problem of finding a perfect matching[16] in a graph $G(U, E)$. The associated polytope P_G is simply the convex hull of all perfect matchings, or more precisely, of all $0 - 1$ vectors in $\mathbb{R}^E$ that are characteristic vectors of perfect matchings. The facets of this polytope were determined by Edmonds [Edm65a], who proved that the following linear program defines P_G. The variables are x_e for every edge $e \in E$, and the inequalities are:

- $\sum_{e \ni u} x_e \leq 1$ for every vertex $u \in U$.

- $\sum_{e \subseteq S} x_e \leq (|S| - 1)/2$ for every *odd* subset $S \subseteq U$ in G.

When G is a bipartite graph, it has no odd cycles, and the second family of inequalities is not needed, f becomes the number of vertices, and thus the polynomial time linear programming algorithms can solve perfect matching in polynomial time for any bipartite graph. However, some other graphs G have exponentially many odd cycles, the polytope above has exponentially many facets, and this method cannot be used. For the perfect matching problem, Edmonds bypassed this issue and discovered a very different polynomial-time algorithm [Edm65b]. However, when formulating standard $\mathcal{NP}$-complete problems (e.g., clique, Hamilton

---

[16]Recall that a perfect matching is a subset of the edges that cover every vertex exactly once.

cycle, traveling salesman) in the same way as linear programs, they all have exponentially many facets, and of course a polynomial-time algorithm for any will imply $\mathcal{P} = \mathcal{NP}$.

Extension of polytopes suggests a general method for reducing the number of facets, by adding some auxiliary variables. A polytope $Q \subseteq \mathbb{R}^{n+n'}$ is called an *extension* of $P \subseteq \mathbb{R}^n$ simply if P is a *projection* of Q, namely, $P = \{x : \exists y, (x, y) \in Q\}$, where $y \in \mathbb{R}^{n'}$. Here the x are the original variables of P, and the y are the new auxiliary variables. Clearly, P is empty iff Q is. Let m be the number of facets of Q. So if we can make $n' + m$ much smaller than the original number of facets f, we win. This possibility is not a pipe dream; $n' + m$ can sometimes be exponentially smaller than f.[17] Can something like this be done for the TSP polytope, as Swart attempted?

Define the *extension complexity* of P, $e(P)$, to be the smallest number of facets m in any polytope Q that extends P (it will automatically turn out that $n' = O(m)$ for this polytope Q, so we need not worry about it). Yannakakis' first brilliant idea was a simple, complete characterization of $e(P)$, and his second was showing how nondeterministic communication complexity implies (nearly tight) lower bounds on $e(P)$. A critical definition is that of a *slack matrix* of a polytope P, which we now give.

Let F and V respectively be the facets and vertices of $P = \{x : Ax \leq b\}$. Then the *slack matrix* S_P of P is an $F \times V$ matrix whose (i, j) entry is simply $b_i - \langle a_i, v_j \rangle$, namely, the distance from the jth vertex to the ith facet. It is crucial to observe that all entries of S_P are nonnegative! The next definition is of the *nonnegative rank* of a nonnegative matrix. It is defined like the usual rank of a matrix, insisting throughout on nonnegativity. Namely, for an $f \times v$ nonnegative matrix S, let $\mathrm{rk}^+(S)$ be the smallest m, such that $S = RT$ for nonnegative matrices R, T of dimensions $f \times m$ and $m \times v$, respectively. With these definitions, we can now state the characterization, sometimes called "Yannakakis' factorization theorem."

Theorem 15.14 [Yan91]. *For every polytope P, $e(P) = \mathrm{rk}^+(S_P)$.*

The proof of this theorem follows directly from the definitions, linear algebra, and the single "convex" fact (called "Farkas' lemma") that if a linear inequality is logically implied by a set of other linear inequalities, then in fact it must be a nonnegative linear combination of these inequalities.

In light of this discussion, we would like to prove (hopefully exponential) lower bounds on $e(P)$ for some of the polytopes mentioned above. What can be said about the function rk^+? First, unlike the usual rank, nonnegative rank is a nasty function, and is $\mathcal{NP}$-hard to compute even on Boolean matrices [Vav09]. Clearly, the usual rank (over the reals) provides a lower bound, namely, $\mathrm{rk}^+(S) \geq \mathrm{rk}(S)$ for every S. But this bound can be extremely weak; for example, there are matrices S with $\mathrm{rk}(S) = 3$ and unbounded $\mathrm{rk}^+(S)$ [Hru12]. Here comes the helpful connection to communication complexity, and to use it, making our matrices Boolean will help.

For a nonnegative matrix S, let $\hat{S}$ be the Boolean matrix, where we replace every positive entry with 1 (and leave the 0s alone). Note that if $\mathrm{rk}^+(S) = m$, then the 1's in $\hat{S}$ can be covered by m monochromatic 1-rectangles. Specifically, if $S = RT$, then the m rectangles $\hat{R}_i \otimes \hat{T}^i$ cover the 1's in $\hat{S}$, where R_i and T^i are respectively the ith row and ith column of R and T. In short, all we need is a polytope P such that $\hat{S}_P$ has no small monochromatic cover.

Yannakakis [Yan91] set all this up, but actually could not fully deliver the goods. He was able to show that Swart's attempts were doomed using an extra property of Swart's construction—symmetry—which when taken into account in this framework enabled him to obtain exponential lower bounds on the relevant "symmetric" extension complexity of the TSP polytope. But he left open the question of proving exponential lower bounds on the general extension complexity, which would completely rule out this linear programming approach to proving $\mathcal{P} = \mathcal{NP}$.

This was finally achieved 25 years later, in the paper of Fiorini et al. [FMP+15]. Let K_n denote the complete graph on n vertices.

Theorem 15.15 [FMP+15]. *The extension complexity of the TSP polytope of K_n is $\exp(n)$.*

We only sketch the high-level ideas of the proof. Rather than studying the slack matrix of the TSP polytope, the proof proceeds indirectly. Given the fact that TSP is an $\mathcal{NP}$-complete problem, Fiorini et al.

---

[17]As an exercise, prove that the convex hull of all *odd* weight n-bit vectors requires $\exp(n)$ facets, and that adding $O(n^2)$ new variables reduces the number of facets to $O(n^2)$.

reason that finding *any* explicit polytope whose slack matrix is hard to cover in the sense above would do. Once one is found, standard $\mathcal{NP}$-completeness reductions would do the job. The difficulty remains to find an appropriate polytope. Their ingenious idea is to use the so-called *cross polytope*. I will not describe it here, but instead describe the properties of its slack matrix S. Both the faces and the vertices of the cross polytope (namely, the rows and columns of S) can be put in $1-1$ correspondence with subsets of $[n]$. If two sets are *disjoint*, then the respective entry of S is 1. If the two sets intersect in a single element, the respective entry of S is 0. What about other pairs? We don't care! Recall again the strong lower bound of Theorem 15.3. It is not hard to see that it implies (and some proofs actually prove it this way) that any cover of the 1's in S requires $\exp(n)$ many rectangles. This provides the required lower bound on $\mathrm{rk}^+(S)$ and hence the theorem.

One can use the same idea to prove exponential lower bounds on the extension complexity of polytopes associated with many other $\mathcal{NP}$-complete problems via reductions. What could not be done this way, and remained intriguing, was to determine the extension complexity of the perfect matching polytope for general graphs discussed in the beginning of this section. This was resolved by Rothvoss [Rot14]. Interestingly, this lower bound bears some similarity to the one in Theorem 15.12, and indeed, this connection was formalized in [Hru12] to show how extension complexity lower bounds can yield formula size lower bounds.

Theorem 15.16 [Rot14]. *The extension complexity of the Perfect Matching polytope of K_n is $\exp(n)$.*

15.2.6 Pseudo-randomness

The application here may look different from all the others mentioned before. So far, we have seen computational lower bounds derived from communication complexity lower bounds. Here I sketch how a construction of a pseudo-random generator can be based on communication complexity. However, it is quite similar in spirit to the other applications, and we leave it to the reader to articulate the similarity (one hint is recalling that pseudo-randomness implies hardness).

Nisan's celebrated space-bounded pseudo-random generator [Nis92] has already been mentioned twice in this book, in Section 8.5 and again in Section 14.1 as Theorem 14.4. This generator was motivated by a space lower bound mentioned at the end of Section 14.2, and both generator construction and proof of pseudo-randomness use it implicitly. A couple of years later Impagliazzo, Nisan, and Wigderson [INW94] described a different pseudo-random generator, for which the proof of pseudo-randomness follows explicitly by a direct reduction to a statement about communication complexity.[18] To describe it, we'll need to recall the notion of an expander graph (see Section 8.7). To suit the communication complexity framework, I define here a *bipartite* expander, for which the expansion property stated is analogous (and more precise) to the property S_1 in Section 8.7 (namely, that between any two subsets of vertices we find about the same number of edges as in a random graph with the same degree). The explicit constructions defined and described in Section 8.7 yield the bipartite expanders we need here.

A *bipartite expander* is a $D = 2^d$-regular bipartite graph $H(A, B; E)$ on two sets of vertices A, B of size $N = 2^n$, such that for every two subsets $A' \subseteq A$, $B' \subseteq B$, we have $||E(A', B')|/dN - |A'||B'|/N^2| \leq \epsilon = 2^{-d/3}$ (where $E(A', B')$ is the set of edges between the two subsets). H is *explicit* if there is a $\mathrm{poly}(n, d)$ time algorithm that for every vertex v and an index $i \in [D]$ outputs the ith neighbor of v in H.

Note that the expansion condition is essentially one about the density of 1's *rectangles* in the adjacency matrix of H relative to their size. Not surprisingly, as communication protocols are essentially partitions of a matrix to rectangles, this property translates immediately to the following communication complexity statement. We use the following notation. For a communication protocol P on n-bit inputs to Alice and Bob, and a distribution μ on the inputs, let's denote by $P(\mu)$ the probability that P outputs 1 when the inputs to the players are chosen at random according to μ.

Lemma 15.17. *Let P be any c-bit communication protocol, and H any bipartite expander with $d = 4c$. Then $|P(U) - P(H)| \leq 2\epsilon$, where U is the uniform distribution, and (abusing notation) H is the distribution picking a random edge from H and giving each player one endpoint of this edge.*

---

[18]Due to this property, [INW94] can generalize the construction and give pseudo-random generators for more general computations.

I will not describe here the construction of the generator, but note that the expanders are used in [INW94] in a similar way to how hash functions are used in the original [Nis92]. The key property for the final generator is the reduction in randomness entailed in using the distribution H as opposed to U. Sampling from U requires $2n$ bits, whereas sampling from H requires only $n + d$ bits. This saving is compounded via recursion in the same way as in [Nis92].

15.3   Interactive information theory and coding theory

The field of communication complexity, besides creating a deep and broad theory with lots of applications as we have seen, has also branched naturally into studying basic issues that were classically in the realm of information and coding theory. These large fields are focused mainly on *information transmission*, namely, *1-way* communication of a *message* held by one party.[19] In contrast, communication complexity studies *information exchange*, namely, *2-way* communication of *adaptive conversations* (e.g., some arbitrary function or relation of both parties' inputs). Asking the classical questions of these fields, like the possibility of compression and tolerance to channel noise, naturally becomes far more challenging in the general *interactive* set-up. This study, which has already led to some beautiful results and more open questions, is becoming a field of its own, at the intersection of information theory and computational complexity. We review below some highlights, first regarding compression of communication protocols, and then error-correcting schemes for them. In both, I explain the similarities and differences of the 1-way and 2-way settings.

Before starting, recall the following standard information-theoretic quantities, defined by Shannon in his seminal paper [Sha48] that created information theory. Let there be random variables A, B, Z on the same finite set. $\mathbb{E}$ denotes expectation. All logarithms are base 2. We will develop further intuition about these quantities as we discuss them below.

The *entropy* of A, intuitively, the uncertainty in A measured in bits, is denoted $H(A)$ and defined by

$$H(A) = -\sum_a \Pr[A = a] \log \Pr[A = a].$$

The *conditional entropy* of A given B (intuitively, how much uncertainty is left in A after B is revealed) is denoted $H(A|B)$ and defined by

$$H(A|B) = \mathbb{E}_b[H(A|B = b)].$$

The mutual information of A and B (intuitively, how much knowing B reveals about A and vice versa) is denoted $I(A; B)$ and is defined by

$$I(A; B) = H(A) - H(A|B) = H(B) - H(B|A) = H(A) + H(B) - H(A, B).$$

All these notions make sense when conditioning on a third variable Z. In particular, the *conditional mutual information* will be of particular relevance to us and is defined by

$$I(A; B|Z) = H(A|Z) - H(A|B, Z).$$

15.3.1   Information complexity, protocol compression, and direct sum

A central complexity measure for communication tasks, besides communication complexity itself, is *information complexity*, introduced by Chakrabarti et al. in the paper [CSWY01], and evolved in [BJKS04] and then [BBCR13], whose definition we shall use here. There are several (related) ways to motivate information complexity. One is as a generalization of Shannon's source coding theorem for 1-way communication to the 2-way setting of communication complexity. Another is by trying to find a more "continuous parameter" than communication complexity, which is always an integer. A third is from the point of view of *amortized* complexity, when one attempts to solve the problem for many instances at once. We will see all these, but let

---

[19]Needless to say, this focus is plenty broad as is. The number of theoretical models and practical settings under which such a basic question is studied is vast and has occupied thousands of researchers for decades and many more in industry; the results are present in the technology we all regularly use.

us motivate the definition from trying to prove lower bounds on (distributional) communication complexity using information-theoretic means.

First, let us consider an arbitrary communication protocol π when applied to a pair of inputs distributed according to a distribution μ. We abuse notation and let (X, Y) denote the input random variables (jointly distributed according to μ). When executed on such random input, π defines another random variable $\Pi = \pi(X, Y)$, the *transcript* or *conversation* between Alice and Bob. Following the intuitive meaning of mutual information, it is clear that Alice, holding X, learns at least $I(Y; \Pi | X)$ bits on average about Bob's input Y from their conversation Π. Similarly, Bob learns $I(X; \Pi | Y)$ on average about X. Thus, the players must exchange at least as many bits, on average, as the sum of these two quantities. Let us formalize this.

For any protocol π and distribution μ, define the *information complexity*[20] $I_\mu(\pi) = I(Y; \Pi | X) + I(X; \Pi | Y)$. Also, let $C_\mu(\pi) = \mathbb{E}_\mu |\pi(X, Y)|$ be the expected length of the communication. Then the argument above can be simply formulated to prove the basic lower bound on the expected length $C_\mu(\pi)$.

Theorem 15.18 [CSWY01, BBCR13]. *For every π, μ, $C_\mu(\pi) \geq I_\mu(\pi)$.*

Clearly, this lower bound extends to collections of protocols. For example, let $\Pi(f, \mu, \epsilon)$ be the set of all deterministic protocols computing f with probability at least $1 - \epsilon$ on input distribution μ. Then the distributional communication complexity $C_{\mu,\epsilon}(f)$ of this task has already been defined as the minimum of $C_\mu(\pi)$ when π ranges over this set $\Pi(f, \mu, \epsilon)$. We similarly define $I_{\mu,\epsilon}(f)$, the information complexity of this task, as the *infimum*[21] $I_\mu(\pi)$ over all π in that set. We then have the following theorem.

Theorem 15.19 [CSWY01, BBCR13]. *For every f, μ, ϵ, $C_{\mu,\epsilon}(f) \geq I_{\mu,\epsilon}(f)$.*

From now on, we will think of $\epsilon > 0$ as tiny (indeed, negligible in other parameters) and will ignore it in our notation. Further, let us fix a distribution μ, and so remove it as well. Finally, it will be useful to think of f as a more general task than just computing a function, and assume that it can be any requirement on the outputs of a protocol (e.g., a relation, or different functions for each player, or even distributions). Everything discussed so far, and that will be discussed later, holds in this generality. We will aim to understand the basic question: How good (or tight) is this lower bound $C(f) \geq I(f)$? We will also ask the same questions for the *amortized* versions of these quantities, which we now define.

Since Shannon's paper [Sha48], a major focus of information theory has been the cost, per input in the limit, of performing the same task many times on independent inputs. In the CS literature, this is known as the *direct sum* problem, which arises naturally for *any* computational model and resource. Denote by f^k the task in which Alice gets $(X_1, X_2, \ldots, X_k)$, and Bob gets $(Y_1, Y_2, \ldots, Y_k)$, with the (X_i, Y_i) independent, each distributed according to μ, and the parties must succeed in performing each of $f(X_i, Y_i)$ with probability at least $1 - \epsilon$ (remember that their messages may use their entire inputs). Let us denote by $\bar{C}(f) = \lim_{k \to \infty} \frac{1}{k} C(f^k)$, and similarly $\bar{I}(f) = \lim_{k \to \infty} \frac{1}{k} I(f^k)$. The paper [CSWY01] examined the basic direct sum problem and the tightness of the obvious lower bound $C \geq \bar{C}$. It introduced information complexity mainly because, for this measure, the two are the same![22]

Theorem 15.20 [CSWY01, BBCR13, BR14]. *For every f, μ, ϵ, $I = \bar{I}$.*

To study the two basic questions (namely, how close the lower bounds $C \geq I$ and $C \geq \bar{C}$ are), let us seek intuition from early examples of communication tasks that were studied in classical information theory. Note that all may be viewed as *compression* results; in all, I naturally captures the *information content* of the problem, and we are trying to get the communication C to be as close to it as possible. Naturally, some of the original results were not stated in this language. We will also add or simplify using hindsight.

First consider the case where Bob has no input (so y can be thought of as empty, or constant). The function to compute is the identity function on Alice's input: $id_A(x) = x$. Thus Alice simply wants to send her input (sampled from some distribution X) to Bob. Because Bob learns x (and Alice can learn nothing

---

[20]This is often called *internal* information complexity, to distinguish it from a related measure, *external* information complexity, capturing the amount of information a protocol reveals to an external observer about the players' inputs.

[21]Being a continuous measure, there are tasks (even simple ones, like taking the AND of two bits) that have an infinite sequence of protocols with better and better information complexity.

[22]Proving this to within a factor of 2 is easy and sufficed for the motivation; the exact result follows from [BR14], which we shall soon discuss.

new), every protocol π must satisfy $I(id_A) = I(X; \Pi) = H(X)$. Shannon [Sha48] proved the following lower bound, and the elegant Huffman coding [Huf52] gives the very nearly matching upper bound.[23] The amortized bound is Shannon's famous *source coding theorem*.

Theorem 15.21 [Sha48, Huf52].

- $H(X) = I(id_A)$.

- $I(id_A) \leq C(id_A) \leq I(id_A) + 1$.

- $I(id_A) = \bar{C}(id_A)$.

Now reconsider the problem: Alice must transmit her input x to Bob, but now in the general situation that Bob does have an input y, and the pair is distributed as (X, Y). In short, they are computing $id_A(x, y) = x$. Using the same reasoning as before, any protocol must give Bob full knowledge of Alice's input. Thus, every protocol π satisfies $H(X|Y, \Pi) = 0$, and so $I(X; \Pi|Y) = H(X|Y)$. Of course, some protocols may give Alice information about Bob's input. At any rate, we have $I(id_A) \geq H(X|Y)$. The amortized case was studied in the well-known paper of Slepian and Wolf [SW73], and the one-shot case by Orlitzky [Orl92]. Note that, unlike Slepian-Wolf, who considered only 1-way communication, Orlitzky considered 2-way communication but showed that it doesn't help for this problem (and so Alice learns nothing).[24]

Theorem 15.22 [SW73, Orl92].

- $H(X|Y) = I(id_A)$.

- $I(id_A) \leq C(id_A) \leq (1 + o(1))I(id_A)$.

- $I(id_A) = \bar{C}(id_A)$.

One final example is the problem $id(x, y) = (x, y)$, namely, the two players must exchange their inputs. This was studied by El Gamal and Orlitsky [EGO84]. They have several results, which we slightly overstate and summarize informally, about the one-shot task.[25] The amortized case of course follows from the Slepian-Wolf Theorem 15.22, applied to id_A and id_B separately. Note that arguing as above, $H(X|Y, \Pi) = H(Y|X, \Pi = 0)$ must hold for any id protocol, and so $I(id) \geq H(X|Y) + H(Y|X)$.

Theorem 15.23 [SW73, Orl92].

- $H(X|Y) + H(Y|X) = I(id)$.

- $I(id) \leq C(id) \leq (1 + o(1))I(id)$ for "almost" all distributions μ.

- $I(id) = \bar{C}(id)$.

Let us try to generalize from these examples. First, consider the amortized case, which seems to be cleaner. In all examples, we have $I(f) = \bar{C}(f)$. Indeed, these equations are often viewed as *operational* definitions of entropy and conditional entropy, giving practical motivation to their abstract mathematical definitions above. What about other tasks f? The important theorem of Braverman and Rao [BR14] shows that we always have equality, thus giving a precise characterization of amortized communication complexity.

Theorem 15.24 [BR14]. *For every task f, $I(f) = \bar{C}(f)$.*

Let me say a few words about the proof. As we have seen in other chapters, picking the right "universal" or "complete" task f would do the job via reduction, which is precisely what they do. This task is abstracted as a joint sampling problem on a pair of distributions that are close in the *KL-divergence* metric. Their

---

[23]I am cheating a bit here, because for Huffman coding, C really denotes "average-case" as opposed to the "worst-case" communication complexity that we are using throughout.

[24]A good example to consider is when Bob's y is a pair of n-bit files, (z_0, z_1), and Alice has one of them. If Bob talks first, it is easy to solve the problem with $\log n + 1$ communication. Can you do the same when only Alice talks? Hint: hashing!

[25]A good example to consider here is that x is a random n-bit file, and y is another random file differing from it in some random set of coordinates of size at most s.

communication-efficient protocol for this problem (which *does* require 2-way communication) may be viewed as a generalization and strengthening[26] of Theorem 15.22 (which does not).

We now return to the "one-shot" compression problem, namely, how tight $C \geq I$ is in general. The first paper to directly address this general question; develop general techniques for protocol compression; and in particular, prove the best general compression result known so far was by Barak et al. [BBCR13]. They focus on compressing a given protocol, which of course implies compression results for general tasks. This can be informally defined as follows. Fix an arbitrary input distribution[27] and a protocol π. We are looking for another protocol π', which (with probability $1 - \epsilon$, as usual) will compute the transcript $\pi(X, Y)$. Hopefully, π' will be compressed (namely, will use less communication than π), perhaps as little as or close to its information complexity (which is the minimum possible). Another desired property is that the computations of Alice and Bob in π' are essentially as efficient as those in π. This efficiency requirement holds for all known simulations.

Theorem 15.25 [BBCR13]. *For every protocol π, there exists another protocol π' such that $C(\pi') \leq \sqrt{C(\pi) \cdot I(\pi)}(\log C(\pi))^{O(1)}$.*

This result shows that one can always compress communication of any protocol roughly to the geometric mean of its original C and I. How good is this result? One can construct artificial protocols (and there even exist natural ones) where C is far greater than I, unboundedly so. In such cases, communication reduces by a square root but does not approach the information complexity. Thus, it would be nice to obtain a compression result that depends only on I. Such a compression was discovered by Braverman [Bra15].

Theorem 15.26 [Bra15]. *For every protocol π, there exists another protocol π' such that $C(\pi') \leq \exp(I(\pi))$.*

Are there much better compressions possible? Ganor, Kol, and Raz [GKR14, GKR15] proved that Theorem 15.26 cannot be improved, not only for individual protocols, but for computing certain tasks as well.

Theorem 15.27 [GKR14, GKR15]. *For every integer m, there exists a Boolean function f such that $I(f) = O(m)$ but $C(f) \geq 2^m$.*

We can now interpret these results in light of the characterization $\bar{C} = I$ of the amortized communication complexity. These represent significant progress on the very old *direct sum* problem in communication complexity. When this problem was raised, in the 1980s, it was believed to be possible that solving any problem k times requires roughly a k-fold increase in the communication cost:[28] $\bar{C}(f) = \Omega(C(f))$. However, no nontrivial upper or lower bounds were found for decades. The information complexity approach implies the following. Combining Theorems 15.26 and 15.27, we have one tight bound.

Theorem 15.28 [BR14, Bra15, GKR15].

- *For every communication task f, $\bar{C}(f) \geq \Omega(\log C(f))$.*

- *For some Boolean functions f, $\bar{C}(f) \leq O(\log C(f))$.*

Also, Theorem 15.25 implies that some multiplicative cost must be incurred in amortization, which is better stated when we explicitly take into account the number of instances solved (and suppress logarithmic factors).

Theorem 15.29 [BBCR13, BR14]. *For every task f and integer k,*
$C(f^k) \geq \Omega(\sqrt{k}C(f))$.

15.3.2 Error correction of interactive communication

We now turn to dealing with *noise* on the communication channel, and how to make communication reliable despite it. We focus on the most typical noise model, namely *bit-flips*. The major idea of battling noise using

---

[26]Especially with respect to the convergence rate to the limit.

[27]We survey here only results regarding general input distributions μ. Much more is known for restricted families of distributions, in particular, when X and Y are independent.

[28]Indeed, the fact that this bound holds for the monotone KW-relations mentioned above was key for proving some superpolynomial lower bounds for monotone formulas in [KRW95].

error-correcting codes was introduced in two papers by Shannon [Sha48] and Hamming [Ham50]. Shannon studied *random errors*, and Hamming *adversarial errors*; we'll discuss both, first in the 1-way communication and then in the interactive setting. An excellent detailed survey of the material summarized in this section is [Gel15].

Assume that Alice wants to send Bob an n-bit message x. However, assume that every bit sent across their communication channel may be flipped, independently of all others, with probability $\leq p$. The parameter p is the maximum "noise rate" per bit. The idea of error-correcting codes is to send an *encoding* of x, which has some redundancy to counter the noise. Formally, an error-correcting code is a function $C : \{0,1\}^n \to \{0,1\}^m$. The ratio n/m, capturing the redundancy of C, is called the *rate* of the code C, denoted $R(C)$.

The code C tolerates *adversarial* noise p if for every n-bit message x, and for m-bit sequence z, which differs in at most pm coordinates from $C(x)$, the original message x can be uniquely *decoded* from z. As Hamming points out, this is possible if and only if the *Hamming distance* $d_H(C(x), C(x')) > 2pm$ for every different message $x \neq x'$. Note that this forces $p < 1/4$.

The code C tolerates *random* noise p if for every n-bit message x, if we independently flip every bit of $C(x)$ with probability $\leq p$ to obtain z, the message x is uniquely decoded from z almost surely.[29] Note that this forces $p < 1/2$. Shannon determined *precisely* the redundancy needed for every noise rate p. Indeed, he proved, in one of the earliest applications of the *probabilistic method*, that a *random code* of that rate will do so. Let $H(p)$ denote the binary entropy function.

Theorem 15.30 [Sha48]. *For every $p < 1/2$, and every $R > 1 - H(p)$, there is a code C of rate $R(C) = R$ that tolerates random noise p. Moreover, no code C of rate $R(C) < 1 - H(p)$ tolerates random noise p.*

Following these papers, Gilbert [Gil52] and Varshamov [Var57] proved that a constant rate is also achievable for adversarial errors, although the exact rate needed is still an open question.

Theorem 15.31 [Gil52, Var57]. *For every $p < 1/4$, there is a code C of rate $R(C) > 1 - H(2p)$ that tolerates adversarial noise p.*

In particular, a constant size blow-up $m = O(n)$ is sufficient redundancy to tolerate the maximum possible noise level in both models. Of course, this is hardly sufficient for practice, as the codes above are not given explicitly, and the encoding and decoding procedures they suggest are highly inefficient. A long sequence of works has finally led to explicit codes with extremely efficient encoding and decoding algorithms in both models. Major achievements were Spielman's [Spi95] constant rate code with linear time encoding and decoding in the adversarial noise setting, and Arıkan's [Arı09] *polar codes*, which achieve Shannon's bound with near-linear time encoding and decoding. In short, despite many remaining technical questions regarding these and other parameters, this basic problem of protecting 1-way communication against errors is well understood.

Now suppose that we change the problem from Alice sending an n-bit *message* to Bob across that noisy channel, to Alice and Bob having an n-bit *conversation*. The main difference to stress between these tasks is that a conversation is *adaptive*. Bob's response to Alice's first bit may depend on its value. And this dependence diverges exponentially as the conversation proceeds. There are many examples that illustrate the devastating effect of even a little noise on adaptive conversations. Imagine, for example, a game of "20 questions" (or more generally, binary search), where the first answer is flipped; the search proceeds in the entirely wrong part of the space. Similarly, suppose the parties play a game of chess, and Alice's first move, say, "e4," is received on Bob's side as "d4." The rest of the game will be nonsensical.

So, suppose we want to protect against noise via error-correcting codes, as in the 1-way case. The main difference to stress between these two settings is that in the 1-way case, the entire message is there, held by one party, which can encode it *at the start*. But now neither Bob nor Alice know the conversation, as it is evolving. The best they can do is encode each new bit they send, possibly with the history of the conversation so far. But this looks like it might have little value, as one key element in all classical error-correcting codes is the dependence of each output bit in $C(x)$ on a large number of bits from x (indeed, a constant fraction).

The analog of Shannon's work for error correction in this interactive setting is the sequence of seminal papers by Schulman [Sch92, Sch93, Sch96] in the 1990s. In these works, he both formulated the problem

---

[29] For Shannon, "almost surely" meant with probability $1 - \exp(-n)$.

and proposed the first solution, remarkably showing that in the interactive setting, almost nothing is lost: One can protect against constant error rate p, even adversarial, paying only a constant factor overhead in the length of the communication! Schulman defined error-correcting *protocols*, and what it means for such protocols to tolerate both adversarial and random errors at bit rate p. The *rate* of such a protocol is, as in the classical case, the ratio of the number of bits communicated in the original (noiseless) protocol and the number of communicated bits in the error-tolerant one. Error-correcting protocols can have complex structure; here we discuss only one elegant notion from Schulman's work that many error-correcting protocols are based on.

A central notion Schulman introduced was an interactive analog to classical error-correcting codes, which he called *tree code*. As is forced by the players' limited knowledge, define $C : \{0,1\}^n \to \Sigma^n$ to be a tree code if for every x and index $i \in [n]$, $C(x)_i$ depends only on the first i bits of x. Here Σ is a finite alphabet (e.g., if $\Sigma = \{0,1\}^c$, then the output length of C is $m = cn$). The rate C of a tree code is, as before, the ratio of input to output length, so $R(C) = 1/|\Sigma|$.

A central measure of quality of tree codes, generalizing the (normalized) Hamming distance for classical code, captures how much encodings differ after they first diverge. More precisely, a tree code C has *relative distance* $\delta = \delta(C)$ if for every two inputs $x \neq x'$, if $C(x) = uw$ and $C(x') = uw'$, then $d_H(w, w') \geq \delta|w|$, where u is their longest common prefix (and so, the first bits of w and w' differ).

Schulman's main results were the existence of tree codes with constant rate and distance, and a protocol allowing the parties to use such codes in interactive error correction, even for adversarial errors. Note that, unlike for 1-way communication, utilizing a tree code for decoding is highly nontrivial, as the players have to catch errors and correct them during the protocol (rather than only at the end), sometimes re-encoding parts where too many errors occurred. Thus unlike the 1-way case, the input x to the tree code C is far from being the original, non-noisy conversation!

Theorem 15.32 [Sch96]. *There exist tree codes C with $R(C) = O(1)$, $\delta(C) = \Omega(1)$. Consequently, there are error-correcting protocols that tolerate adversarial (and hence also random) error rate $p < 1/240$.*

As in classical error correction, the proof is via the probabilistic method, and so the code is not explicit, and the encoding and decoding are inefficient. This was subsequently remedied, and a sequence of works led to highly efficient, probabilistic error-correction protocols that can tolerate constant adversarial error rate. The best known such protocols are in [BKN14, GH14] (with the last one achieving the best possible error rate, $p < 1/4$). No analogous deterministic protocols are known.

Finally, let us address the random noise model, in which the classical 1-way case achieved a particularly satisfying complete understanding of the trade-off between noise rate and the rate of the error-correcting protocol in Theorem 15.30. Recall that for noise rate p, the optimal code rate is $1 - H(p)$. In the interactive case, what we know is far less precise. However, it is known that such a rate cannot be achieved, at least for very small values of p. The results are somewhat sensitive to the exact communication model. One weak analog of Shannon's theorem was given by Kol and Raz [KR13] and another by Haeupler [Hae14]. Ignoring logarithmic factors and the model details, they can be summarized informally as follows.

Theorem 15.33 [KR13, Hae14]. *The best possible rate of an error-correcting protocol that tolerates random noise rate p is $1 - \Theta(\sqrt{H(p)})$.*

Note that for small p, $H(p) \approx p \log 1/p$, which demonstrates that interactive error correction is more costly than non-interactive error correction. In this regime (and ignoring logarithmic factors), 1-way communication needs to add to an n-bit message about pn extra bits of redundancy, whereas interactive communication needs to add to an n-bit communication about $\sqrt{p}n$ bits of redundancy.

Determining the interactive trade-off more precisely is a very interesting open problem, especially for fixed p (about which Theorem 15.33 says little). Also, in these results too, the protocols are probabilistic. Obtaining deterministic protocols that are as efficient as probabilistic ones—for both adversarial and random noise—is another important question (see the current state-of-art in [GHK+16]). The best approach to the last question, and perhaps the most elegant open problem of this theory, is the construction of explicit tree codes with constant rate and distance.

Open Problem 15.34. Construct explicit, efficiently encodable and decodable tree codes of constant rate

and constant relative distance.

A beautiful construction, whose correctness rests on an unproven conjecture regarding exponential sums, is given in [MS14b].

16 On-line algorithms: Coping with an unknown future

Is hindsight really 20/20, as the saying goes? What exactly is the power of clairvoyance? Here are a few concrete examples from people's lives, encompassing the need to make periodic decisions without knowing the future. They illustrate the need for the models, algorithms, and analysis we discuss in this chapter.

- **Investment:** You own some portfolio of stocks. Every day (or month, or year), given the prices of the stocks, you make a decision to buy and sell some. How should you choose?

- **Gym:** You go on a whim every so often to the gym (or the theater). When you go tomorrow, should you buy a yearly subscription for $500, or pay only $50 for a single visit?

- **Dating:** You seek a lifetime mate and are in a relationship. Should you continue dating that person, or terminate it in the hope of finding a more suitable individual?

- **Memory:** Some psychologists and neurologists believe that our "working memory" can hold at any one time only 7 (well, some say 4 or 5) different things (concepts, ideas, facts). As your environment changes (you start driving, or meet intellectuals, or sit down for dinner), you subconsciously replace some of them with others—how does your brain decide what to discard and what to upload?

- **Taxi:** You are the taxi dispatcher, and a new request arrives to take someone from A to B. Which of the available taxis should you send?

In all of these examples, a sequence of "events" (requests, facts, etc.) is arriving one at a time, and each requires making a decision. Each decision has a cost/benefit associated with it (which depends on your current state), and it also changes the state you are in. Algorithms facing such situations are called *on-line* algorithms. While the input arrival structure is similar in spirit to the streaming algorithms we saw in Chapter 14, here the task at hand is far more general than saving memory. Indeed, no limit is put on the computational resources required by the decision maker, and the model is a purely information-theoretic one. It isolates only the central aspect: What is the best course of action, as each signal arrives, when future signals are unknown? As can be seen (and imagined) from the variety of the examples, this is an extremely general and important problem, and we will only touch on some basic aspects and examples. A comprehensive book on the subject is [BEY05], and a more recent book is [Haz16]. These and other sources cited here explain the connections of this important area to game theory (and strategies for playing well), convex optimization, learning theory, inductive inference, and more.

The most basic question is: What is a good way to model the quality of an on-line algorithm? A bold answer called *competitive analysis* was proposed by Sleator and Tarjan [ST85]. In contrast to many previous studies, they advocate completely ignoring any knowledge of a potential "prior distribution" about future events. Instead, they suggest comparing the performance of the on-line algorithm on each and every input sequence to that of the best algorithm with hindsight; an optimal "off-line" (clairvoyant) algorithm that knows the full input sequence *before* making any decision. Roughly speaking, an on-line algorithm is called "c-competitive" if for *every* possible input sequence, the cost of the on-line algorithm is within a factor c of the cost of the optimal off-line algorithm. An algorithm is *competitive* if it is c-competitive for some finite c. The boldness in this definition is manifest in the suggestion that a finite c may be achieved, which does not depend of the input (in particular, on the input length). Intuition, and perhaps life experience, suggest that knowing the future should give enormous hindsight power in situations like the examples above. As it happens, this intuition is often wrong! Starting with this seminal paper, which powerfully demonstrated this possibility, numerous examples of scenarios of competitive on-line algorithms were found. I will give a couple of examples (and one non-example) of this surprising phenomenon, but I will first define the relevant terms more precisely.

Let us consider one very general definition (introduced and named *request-answer games* in [BDBK$^+$94]), of which most specific scenarios and models of the field are special cases. For any sequence $z = z_1, z_2, \ldots$ and any integer t smaller than its length, let z_t be the tth element of the sequence, and z^t be the tth prefix of z, namely, $z_1, z_2, \ldots, z_t$.

An *on-line problem* is specified by a set E of events, a set D of decisions, and a family of cost functions $C = \{C_t : E^t \times D^t \to \mathbb{R}\}$ for every integer t. The input to an on-line problem is a sequence $e = e_1, e_2, \ldots, e_T \in E^T$. So, each $e_t \in E$ is the event at time t. The total time (= number of events) T is arbitrary and unknown to the algorithm (and as we shall see, may be thought of as infinite if desired).

An *on-line algorithm* A is a sequence of functions $A_t : E^t \to D$, specifying the next decision given the input so far. Thus it is unambiguous for every t to denote by $A(e^t) = A_1(e^1), A_2(e^2), \ldots, A_t(e^t) \in D^t$ the sequence of t decisions made in the first t time steps by A. The *cost* A incurs in step t is $C_t(e^t, A(e^t))$, and $C_A(e)$ denotes the *total cost* incurred by A over all decisions, namely, $C_A(e) = \sum_{t=1}^{T} C_t(e^t, A(e^t))$.

The *off-line* (or optimal) cost of a sequence e is easy to describe—it is simply the minimum cost incurred by the best sequence of decisions. Namely, $OPT(e) = \min_d \sum_{t=1}^{T} C_t(e^t, d^t)$, where the minimum is taken over all possible $d \in D^T$ (d may be viewed as the actions of a hypothetical off-line algorithm that sees the entire input e in advance).

An on-line algorithm is said to be "*c*-competitive" (for a given on-line problem defined by (E, D, C)) if there is a universal constant[1] M such that for every sequence e (of every finite length), we have

$$C_A(e) \leq c \cdot OPT(e) + M.$$

Sometimes we say that A has *competitive ratio* c. Two simple observations follow. First, if A is c-competitive, no other algorithm does better by a factor of more than c. Furthermore, this statement holds not only for the final input sequence but also for every prefix of it.

Let us start with a simple example of an on-line problem for which there is no competitive algorithm. Assume $E = D = \{0, 1\}$. The problem is to guess the next bit in the sequence. Thus, regardless of the past, you pay (say) \$1 if it is guessed incorrectly and pay nothing otherwise. In symbols, $C_t(e, d) = e_t \oplus d_{t-1}$ for any $t > 1$ and any two sequences d, e of length t. It is obvious that $OPT(e) = 0$ for every e (simply by choosing $d = e$). However, it is clear that for every (deterministic) algorithm A and for every length T, there is a sequence e of length T for which $C_A(e) = T - 1$. Simply pick e_1 arbitrarily, and subsequent e_t values as the complement of what the algorithm predicts, namely, $e_t = 1 - A_{t-1}(e^{t-1})$. Thus, A does not have a finite competitive ratio.[2] This example suggests a very useful view of the competitive analysis definition: The sequence e may be viewed as generated by an adversary, one symbol at a time, having full information about the algorithm A.

We now turn to two very general examples where nontrivial competitive on-line algorithms do exist! These will formalize some of the informal examples mentioned above. Note that some of them are quite simple, and it would be nice for you to try to work them out yourself. For instance, find a 2-competitive algorithm for the gym problem (a special case of the-all-too familiar "buy or rent" problem).

16.1 Paging, caching, and the k-server problem

Consider the following formalization of the "working memory" example above, which actually is quite practical in computer memory systems that are organized hierarchically (as our brain may be organized as well). There is fast memory, or *cache*, which can hold k data items, and a much larger slow memory. The event set E is simply all data items, an arbitrary k of which are initially in the cache and the rest in slow memory. Requests of the system to access data items arrive (this is the input sequence e). If the requested item e_t is in cache, no decision need be made, and no cost is incurred (due to the fast retrieval from cache). However, if the requested item e_t is not in the cache, it must be moved there, and to make room for it, a decision must be made as to which item from the cache is to be moved to slow memory. Regardless of which item is removed, the cost is \$1 (i.e., paying for the time to access slow memory). This problem is called the *k-paging* problem.

It is clear that an adversary can create a request sequence that will make any on-line algorithm pay \$1 at every step, simply by requesting items outside the cache. But note that in such a case, even an

---

[1]In some contexts, M is allowed to grow, but it should be kept asymptotically smaller than OPT.

[2]With the natural extension of on-line problems and competitive analysis to allow randomness, a similar argument can show that even a probabilistic algorithm A cannot achieve a competitive ratio better than $T/2$.

off-line algorithm will have to access slow memory every so often. The question is: Does this problem have a competitive algorithm? The competitive ratio can depend on k but not on the length of the sequence. Think about it before reading on!

A natural heuristic (which our brain may be using, too) is what is known as the "least recently used" (LRU) algorithm. This algorithm simply always discards, when the cache fills up, the item in the cache that has been last requested[3]. This algorithm clearly fits the general model above. The analysis of this algorithm is one of the initial examples in the Sleator-Tarjan paper [ST85], where they determine precisely its quality (in much more generality than we state).

Theorem 16.1 [ST85]. *The algorithm LRU is k-competitive for the paging problem.*
Moreover, no on-line algorithm for this problem has a competitive ratio smaller than k.

Let us now generalize the setting above quite a bit, to the famous *k-server* problem, introduced in [MMS90]. It may be viewed as a concrete model for the taxi problem mentioned at the start of the chapter, and for many others with the following structure. The requests are points in some metric space, there is a limited number (k) of resources to handle requests, and service cost is derived from the distance traveled to service. Formally, fix a metric space $M = (E, dist)$ so the requests set E is simply points in the metric space, and $dist$ is a distance function between pairs of points (satisfying the triangle inequality). The k "servers" initially reside on some points of M. Now a sequence of requests (namely, points of M) arrive one at a time, and your job is to decide which of the k servers to send there. The cost is the distance traveled by that server from its current location to the new request point.

It should be clear that this paging/caching problem is the very special case in which the metric space is *uniform*, namely, the distance between every pair of points is the same (say, 1). Can we have a competitive algorithm in this general setting? A positive answer was conjectured in [MMS90]; indeed, the authors conjectured a competitive ratio of k is possible, as in the uniform case. A major result in the field, by Koutsoupias and Papadimitriou [KP95], comes very close to confirming this conjecture with their ingenious analysis of the so-called work function algorithm.

Theorem 16.2 [KP95]. *The work function algorithm is $(2k - 1)$-competitive for the k-server problem.*

It will not surprise the reader that also in the on-line setting, randomness can play a crucial role, and that the model extends to allow probabilistic algorithms that outperform deterministic ones. For example, there is a probabilistic algorithm (that as usual is correct for every input, with high probability over its random coin tosses) for the k-paging problem that is $O(\log k)$-competitive [FKL+91][4]. As we know that k is the best possible competitive ratio even for the paging problem, we see that randomness can be provably exponentially more powerful in this parameter. An important source of this power comes from the fact that now the adversary generating the input sequence knows the algorithm but does not know the random coin tosses in advance. Subtle issues regarding the interaction of the adversary and probabilistic on-line algorithms (and models capturing them) are studied in [BDBK+94]. These models exhibit natural situations in which randomness can enhance on-line algorithms by at most a polynomial factor.

16.2 Expert advice, portfolio management, repeated games, and the multiplicative weights algorithm

Finally we get to questions about predicting the future that almost everyone is concerned with on a daily basis, like which weather channel to trust and which financial expert to listen to. Amazingly enough, in very general situations, you can do practically as well as the best expert *in hindsight* with only the knowledge of past performance! If you are wondering why such an astounding possibility is not used by everyone to do as well on the stock market as legendary investors like Warren Buffet, well, it is a good question, and there are many answers. Read the theoretical results below, and you can decide for yourself whether to try them at home. They are certainly in extensive use in numerous applications, including financial investments.

---

[3]Practically one can, e.g., keep the items in the cache in an ordered list, placing each newly requested item at the start of the list, and always discarding the last one when needed.

[4]A possible (poly log k)-competitive probabilistic algorithm for the general k-server problem remains open. Major recent progress on this problem can be found in [BCL+18].

Note outright that we are changing the model! We will now be comparing an on-line algorithm to the best one from a restricted family of off-line algorithms (the experts), as opposed to the best off-line algorithm. The reason is simple. In these settings, typically there can be no competitive algorithm in the sense of Section 16.1, and this new, more limited setting of comparing to the best expert in hindsight is still very general.

The key to many of these surprising results is a single, extremely important "meta-algorithm" called the *multiplicative weights update* algorithm, which may be viewed as a smooth way of taking past information into account when making future decisions. Thus it may be viewed as a learning algorithm, and certainly arises in machine learning (which we will discuss shortly). But again, in the spirit of this chapter, no assumptions are made about future events, and they can be determined by an adversary who knows the algorithm. Variants of this algorithm were discovered in diverse application areas by many people. It was even suggested that this algorithm was independently "discovered" by nature, and that it naturally occurs in evolution [CLPV13, MP15]! It is certainly simple enough to be implemented in a distributed fashion by simple organisms and possibly even by genes. This algorithm made an appearance in Section 8.8, in the completely different context of pseudo-randomness; you should compare it with what we'll discuss here. An excellent survey of the many incarnations and uses of this algorithm is [AHK12].

I describe a simple variant of this algorithm, called the *weighted majority* algorithm of Littlestone and Warmuth [LW94], which is designed to handle binary predictions.[5] For example, will it rain or not tomorrow? Or, will the price of a given stock go up or down tomorrow? This algorithm will suggest a method to aggregate the "advice" of k different "experts" about such events into a decision, which over time, *on every sequence*, will perform nearly as well as the predictions of the best expert on that sequence. Let us formalize the on-line problem at hand and then discuss the algorithm and its performance analysis.

The events E are pairs (b, v), where $b \in \{-1, 1\}$ and $v \in \{-1, 1\}^k$, which should be understood as follows. The bit b is a fact, about the reality at this time step (e.g., did a particular stock price go up or down today). The vector v is the opinions of k different experts about reality in the next time step (e.g., whether that stock price will increase or decrease tomorrow). At this point, the algorithm makes its decision, namely, a 1-bit prediction about tomorrow. On the next step, reality is revealed, and the algorithm learns which predictions (its own and the experts') were correct and which were not. The cost of a wrong decision, to the algorithm and to an expert, is (say) $1. Thus, we will be counting mistakes (or wrong predictions), and the goal of the algorithm is to minimize these in comparison with mistakes of the best (in hindsight) expert among the given k. Note that the expert who made the fewest mistakes can change over time.

Now let us describe the weighted majority algorithm, which as mentioned, can be viewed as a simple version of multiplicative weight updates. The updates will be to an evolving estimate of our trust in the different experts. Initially, we trust them all equally, and so assign to each a weight 1. As we observe some experts making mistakes, our trust in them will decrease, *multiplicatively*, by a constant factor. Our decision on how to aggregate their predictions will be simply by a weighted majority according to their current weights. Let me be a bit more specific.

Let $w_t(i)$ denote the weight of the ith expert at time t (so for $t = 0$, we have $w_0(i) = 1$ for all $i \in [k]$). Denote by W_t the total weight at time t, namely, $W_t = \sum_i w_t(i)$. Thus $W_0 = k$. At each step, as the current value of b is revealed, we can tell which experts were right and which were wrong in predicting it in the previous step. We decrease our trust in each particular expert who makes a mistake as follows. If expert i was wrong in predicting in step t, then we set $w_t(i) = w_{t-1}(i)(1-\epsilon)$. Note that the update is multiplicative, and the parameter ϵ often has to be chosen carefully to trade off the speed of learning and the volatility of the predictions. It is good to think of ϵ as a small constant, like .01. Finally, the algorithm predicts the weighted majority to the predictions $v_t(i)$ of the experts with the current weights $w_t(i)$. More precisely, the algorithm predicts the *sign* of $\sum_i w_t(i)v_t(i)$. This algorithm is nearly 2-competitive (with respect to the best expert).

Theorem 16.3 [LW94]. *For any t, let M_t be the number of mistakes made so far by the weighted majority*

---

[5]This algorithm follows similar ones originally developed for *boosting* in computational learning theory, which we discuss in Chapter 17.

algorithm, and let $m_t(i)$ be the number of mistakes made so far by the ith expert. Then for every i,

$$M_t \leq 2(1 + \epsilon)m_t(i) + O((\log k)/\epsilon).$$

The analysis is quite simple and follows the intuition of the algorithm. The following two claims relate the weights and mistakes of both the experts and the algorithm, and follow by induction on t. First, for any expert i, its weight decreases by $(1 - \epsilon)$ with every mistake, and so at time t it is exactly $w_t(i) = (1 - \epsilon)^{m_t(i)}$. As $W_t = \sum_i w_t(i)$, we have that for every i, $W_t \geq w_t(i) = (1 - \epsilon)^{m_t(i)}$. Second, when the algorithm makes a mistake, experts of total weight $W_t/2$ must have made a mistake by the weighted majority rule, and (as $W_0 = k$) we have that $W_t \leq k(1 - \epsilon/2)^{M_t}$. Combining the two bounds on W_t and taking logarithms proves the theorem.

A natural question is whether the competitive ratio of 2 (as surprising and good as it may be) is the best possible. Perhaps even more surprisingly, with the help of randomization, we can approach a competitive ratio of 1, namely, make almost as few mistakes as the best expert in hindsight. This is shown in the same paper of Littlestone and Warmuth, which also suggests the randomized weighted majority algorithm; it updates trust in experts exactly as in the deterministic version and makes only the decision probabilistic. Instead of a weighted majority, the algorithm simply follows the prediction of the ith expert with probability $w_t(i)/W_t$. This happens to smooth out the worst case in the above analysis and yields a nearly 1-competitive algorithm, in *expectation*.

Theorem 16.4 [LW94]. *For any t, let M_t (which is now a random variable) be the number of mistakes made so far by the randomized weighted majority algorithm, and let $m_t(i)$ be (as before) the number of mistakes made so far by the ith expert. Then for every i, the expected number of mistakes of the algorithm is bounded by*

$$\mathbb{E}[M_t] \leq (1 + \epsilon)m_t(i) + O((\log k)/\epsilon).$$

The performance of on-line algorithms is often expressed in terms of *regret*, namely, the largest gap between the performance of the on-line algorithm and the best performance in hindsight. Let us present the last result in this language. Let $i*$ be the expert with the minimum number of mistakes in some number T of rounds. Thus the (expected) regret $\mathbb{E}[M_T] - m_T(i*)$ is bounded by $\epsilon m_T(i*) + O((\log k)/\epsilon)$. Using the trivial bound $m_T(i*) \leq T$, and choosing ϵ (which can be set at will) to balance the two terms in the bound, one sees that after any number T of steps, the regret is bounded by

$$\mathbb{E}[M_T] - m_T(i*) \leq O(\sqrt{T \log k}).$$

Thus, for very large T, the average regret per step is only $O(1/\sqrt{T})$ (even though it could be 1). This dependence on T is the best possible.

It is not hard to imagine how to generalize the algorithm from this binary decision setting to one in which decisions and costs are continuous, say, in the bounded interval $[0, 1]$ instead of the binary $\{-1, 1\}$. The updates then depend on the magnitude of the error made by an expert, and if that loss is $g \in [0, 1]$, then the algorithm will reduce its weight by a factor $(1 - \epsilon)^g$. The probabilistic algorithm will choose as before to follow an expert picked at random with probability proportional to its weight. The same analysis shows that the cost to the probabilistic algorithm will be as close as we want to the cost to the best expert in hindsight. One beautiful application of this continuous generalization is to playing *repeated games*, which was discovered by Freund and Schapire [FS99], and we briefly describe next.

A natural setting in which we are attempting to do well against a sequence of adversarial moves is of course a *game*. Consider the familiar zero-sum game setting from game theory (think, e.g., of rock-paper-scissors). A real matrix M describes a 2-player (full information) zero-sum game as follows. The Row player picks a row i of M, the Column player *simultaneously* picks a column j of M. The gain of the Row player, which is equal to the loss of the Column player, is $M(i, j)$. Playing such games well has been understood for nearly a century, with von Neumann's discovery of the minimax theorem, which determines the *value* of the game (the best possible outcome for both players). Linear programming provides a polynomial-time algorithm computing the optimal mixed strategies for both players, which achieve the game value.

Does this understanding kill the subject? Far from it, and the following questions were already being considered in game theory in the 1950s. Suppose the players don't know M and are only told their loss/gain after playing. Suppose M is too large for linear programming to be efficient enough. Suppose your opponent plays suboptimally—can you gain more than the value of the game? If the game is played once, there is nothing much one can do with these questions, but if the players play the same game *repeatedly* (e.g., two competitors repeatedly setting the prices of their products), the on-line setting gives this problem a new life. Can you see how to use the algorithm above to play asymptotically in the best possible way against *any* opponent? Let's assume for now that M is known. The idea is to use the weighted majority algorithm above. Consider your pure actions (e.g., the different rows if you are the Row player) as your experts. Each round, as you learn your opponent's move, reveals the gain/loss of each of your own choices/experts. This allows updating the weights, which serve as your mixed distribution for the next action. This play achieves essentially what can be achieved by the best pure strategy against the adversary, and so it does at least as well as the value of the game. Moreover, as explained in [FS99], this algorithm also leads to a simple new proof of von Neumann's minimax theorem. How about playing when M is not known, and only the payoffs are revealed at every step? As it happens, there is a variant of this algorithm that also works (with somewhat worse performance) in this case.

Let us conclude with money, which is probably why you have stuck with this chapter to the end. We now discuss the *Portfolio Management* problem. There are k stocks in which you have initially invested some amount (say, \$1,000), distributed according to a vector $p = (p_1, p_2, \ldots, p_k)$, where a fraction p_i is invested in the ith stock. This vector p describes your portfolio. The values of these stocks vary daily, and the question is how to reinvest their total value. One commonly used approach (good especially for the lazy investor) is simple *rebalancing*—distribute the new value again according to p. Of course, the question is: which p will yield the best performance over time? Just to show how this simple strategy can yield exponential earnings in a volatile market, say you invest in Apple and Microsoft. Further assume that over time, the Apple stock remains flat at \$1, while the Microsoft stock alternates between \$$\frac{1}{2}$ on odd days and \$2 on even days. If your portfolio was $p = (\frac{1}{2}, \frac{1}{2})$, a simple calculation will show that rebalancing will grow your wealth *exponentially*, by a factor $(9/8)^{t/2}$ after any even number t of days. Of course, if your portfolio picked only one stock of the two (either $p = (1, 0)$ or $p = (0, 1)$), your wealth will remain essentially the same over time.[6] It is the best of all these choices (with hindsight) that we will compare ourselves to! We wish to design an on-line algorithm, which can select a *different* portfolio every day after seeing the stock values, which can be competitive[7] against the *best fixed* portfolio rebalancing. The seminal paper of Cover [Cov91] suggests this problem, and defined an on-line algorithm as *universal* if, roughly speaking, for every t and $\epsilon > 0$, when the best portfolio achieves growth c^t in t days, then the online algorithm achieves growth $(c - \epsilon)^t$. Amazingly enough, this can be done, and Cover describes and analyzes one such universal algorithm. His algorithm and subsequent ones were exponential in the number of stocks k, until Kalai and Vempala [KV03] found a polynomial-time algorithm with the same performance as Cover's (an even more efficient algorithm was later given by [HAK07]). These algorithms are not obviously of a multiplicative weight update form; however, [HSSW98] shows that this problem fits the general framework and gives such an algorithm for it.

---

[6]Try finding an example where balancing a fixed portfolio will *shrink* your assets exponentially fast.

[7]In a somewhat weaker sense than defined above, which is suitable for situations where exponential growth can be expected.

17 Computational learning theory, AI, and beyond

In this chapter, we face an extremely general modeling problem: how to define, and then design, algorithms that learn from experience and use it to cope with new, different situations that may arise. In this generality, such algorithms must be able to learn to walk and talk like a child, to fly or swim, find food and shelter like a young animal, make up theories from data like a scientist, prove theorems like a mathematician, play musical instruments and compose, lead an army or start up a company, discuss philosophy and emotions, laugh at jokes, have offspring, and so forth. All life forms around us (and especially humans) seem to be born with some basic capacity and drive to learn, and then go through life acquiring experience and using it (to various degrees of success) to survive, thrive, and reproduce. Even when ignoring the huge scientific question of how algorithms performing these tasks evolved in living beings (a question that computer science should play a major part in answering), we can ask the more concrete question of how to design computer systems with some of these capabilities.

As you know, some aspects of this project are already a reality. We live in the era with great advances in "machine learning." Many computer systems have actually *learned*, rather than having been directly programmed, to do some amazing feats, many of which we use already or will soon use. Such systems provide individual recommendations (e.g., of books, movies, and more), feature and content recognition in images, language translation, medical diagnostics and treatment, weather and stock market predictions, and more. Learning programs are now beating humans at Chess and Go, and may well replace data scientists completely, rather than only assisting them in doing "data science" and "knowledge discovery." Self-driving cars are practically around the corner, promising multiple changes in our daily routine. And there are numerous more examples, many generated daily with the *deep learning* revolution. Indeed, in this (possibly the third or fourth) revival of the AI dream (or nightmare), some predict that within this century, most aspects of human intelligence and cognition will be paralleled or surpassed by machines.

We will not discuss most of these issues or speculations here. Much of the current rapid progress is based on *heuristics*, which use amazing computing capabilities, the huge amounts of data available today (and as always, efficient algorithms). Many practical successes are far from understood, and given the massive deployment of these machine-learning heuristics across society, the much needed theoretical understanding will be improved. In this chapter, we will only discuss some initial, concrete models, algorithms, and mathematical results of computational learning that address some aspects of this extremely complex subject. We will specifically discuss what is known about the power and limits of learning of these models. While they are still evolving, and new ones are being introduced, some of the principles and models suggested in these initial works are naturally used in the current work in machine learning, and they may also serve to model natural phenomena from evolution to cognition.

Learning is a big word, loaded with multiple meanings, about which generations of scholars have debated and written voluminously. An often quoted (operational, as opposed to cognitive) definition of learning algorithms was given by Tom Mitchell [Mit97]: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."

In this chapter, we discuss some concrete meanings that can be given to the undefined notions in this definition. Let us start with the "experience," the interaction of the learner with the environment. One can divide the models broadly into two classes[1] with respect to that interaction. The first, *supervised* learning, implicitly assumes the existence of a "teacher" who provides side information about raw data. The teacher can vary from a devoted parent or actual classroom teacher, to any knowledgeable entity (like Internet users labeling their pictures), to an environment, which rewards/punishes living organisms based on their behavior, be it foraging for food or investing in the stock market. The second, *unsupervised* learning, does not assume such a teacher or any side information, and the learner must make sense of the data without such help. Naturally, these situations are much harder to learn from, and we refer the reader to the extensive textbook [Mur12], which presents a wide variety of models and techniques. The textbook [GBC16] focuses on the recently favored meta-model of *deep learning* (which is used in both the supervised and unsupervised

---

[1] And many intermediate ones.

settings). Here we only discuss supervised learning.

The main (general) task we discuss in the category of supervised learning is *classification*, which in various texts is referred to also as *identification*, *concept learning*, and other terms aimed to capture natural ability to *generalize* and extract a rule (or function, or pattern, etc.) from examples. We will completely ignore the numerous philosophical debates, starting in antiquity, about the possibility of learning from examples, about justification for doing so, and everything else that comes under the heading of *the problem of induction* in philosophy.[2]

17.1 Classifying hyperplanes: A motivating example

In thinking about the stylized identification problem below and the following discussion, it may be useful to think of some familiar problems from real life. For example, a small child learning to identify a particular animal (e.g., cat) from a sequence of images labeled by "this is a cat" or "this is not a cat" (possibly provided by a parent). Or alternatively, think of a computer program (like the one used by Amazon and other companies) that is trying to identify a particular reader's taste in books, from a sequence of book descriptions (e.g., title, author, and synopsis) with the bit "I liked this book" or "I did not like this book" (provided by the reader). Or a scientist, who is trying to identify (the relevant signs of) a disease from a sequence of people (represented by a list of physical characteristics) and for each a bit of whether a certain genetic marker is present or absent. Here we replace animals, readers, and diseases from the real-life examples above with *hyperplanes* in high-dimensional Euclidean space, and the task with identifying a particular hyperplane from labeled points in space (this problem is often referred to as *linear classification*). Hyperplanes may seem like an extremely synthetic, simplistic choice of concept to learn, which no one really cares about. As it happens, identifying hyperplanes is extremely important and is much more relevant than you might imagine to identifying animals, literary tastes, diseases, and many other concepts.

Indeed, let us see that the concept of *spam* e-mail can naturally be viewed as a hyperplane (which is how many spam filters represent them). Suppose that the typical e-mail vocabulary is from a specific vocabulary of 1,000 words. Represent that e-mail as a vector $x \in \mathbb{R}^n$ for $n = 1,000$, where the jth entry $x(j)$ represents the number of occurrences of the jth word from the vocabulary.[3] Now, each of us may have our personal signs of spam. For example, if you see "Viagra" once, that's clearly spam. But possibly you can tolerate four occurrences of "pharmaceutical," two of "click here," as well as the combination of two "pharmaceutical" and one "click here." It makes sense to assign weights $h(j)$ to each word, and to determine whether the total weight $\langle h, x \rangle = \sum_j h(j)x(j)$ exceeds a certain threshold value v. Then the hyperplane in question is represented by the linear equation[4] $\langle h, x \rangle = v$, such that e-mail x is deemed *spam* if $\langle h, x \rangle > v$ and *not spam* if $\langle h, x \rangle < v$. Of course, this hyperplane representation of spam is just a (very simple) model, but let's assume it is an accurate model. Now consider how a spam filter will identify your "individual" hyperplane (which depends on your personal taste and tolerance, which perhaps are subconscious values that even you cannot specify) simply from your marking of different e-mails as "this is spam" and "this is not spam." Let us first formalize the problem, and then discuss algorithms for it.

We can make the following two assumptions without loss of generality. Assume that any vector h is a *unit* vector (namely, has Euclidean norm 1), and that $v = 0$ (namely, the hyperplane goes through the origin).[5] Now consider the task of identifying an unknown vector h^* in $\mathbb{R}^n$ representing such a hyperplane in space through the origin. The information given is a (possibly infinite) sequence of pairs (often called *labeled examples*) $(x_1, b_1), (x_2, b_2), (x_3, b_3), \ldots$, where $x_i \in \mathbb{R}^n$ are points, and $b_i = \text{sign}(\langle h^*, x_i \rangle)$ is the "side" (or halfspace) of h^* that the point x_i resides in, namely -1 if the inner product is negative, and $+1$ if positive (and 0 if x_i is on h^*, although we can assume this lucky case never happens).

We will view this (and classification problems in general) as an on-line problem, similar to the ones studied

---

[2]Note that it is sometimes amusing to read how people, who as babies could neither speak nor reason at all, argue in sophisticated language the impossibility of learning or gaining knowledge from examples.

[3]Note that this representation, which again is a very common way to treat text, ignores the order of words and keeps track only of their frequency.

[4]With h being the normal vector to the actual hyperplane in space.

[5]We lose no generality by adding one more dimension.

in Chapter 16, namely, one in which the algorithm observing the sequence should propose a *hypothesis* after seeing every new example.[6]   Equivalently, we seek an algorithm that for every finite sample produces a hypothesis.

Here are two natural (efficient!) algorithms for this classification task.

Perceptron algorithm   This algorithm will produce, on every input pair, a new hypothesis for the value of h^* and will continue doing so indefinitely, judiciously modifying its last hypothesis if it is inconsistent with the next labeled example. We start by setting $h^0 = 0$, and after the first $t - 1 \geq 0$ input pairs have been processed, with h^{t-1} being the last hypothesis, proceed as follows on the next input pair (x_t, b_t): Do nothing if consistent, and tilt it toward x_t if not. More precisely, we set h^t as follows:

- **Correct classification:** If $b_t = \text{sign}(\langle h^{t-1}, x_t \rangle)$, set $h^t = h^{t-1}$.

- **Incorrect classification:** If $b_t \neq \text{sign}(\langle h^{t-1}, x_t \rangle)$, set $h^t = h^{t-1} + b_t x_t / \|x_t\|$.

The perceptron algorithm was invented by Rosenblatt [Ros58] and analyzed soon afterward by Novikoff [Nov62] (numerous subsequent improvements and generalizations of this analysis followed: see the survey [MR13]). I will describe the analysis later on, in Theorem 17.1 (the interested reader may jump ahead and then return here).

Let us now turn to a possibly simpler and more natural (albeit somewhat less efficient) algorithm that uses linear programming.[7]

Linear programming   Every finite number of labeled examples naturally defines a system of linear inequalities satisfied by h^*. More precisely, assume that we have s (a parameter) labeled examples. For each such input pair (x_i, b_i) with $i \in [s]$, write a linear inequality $b_i \langle h, x_i \rangle \geq 0$, whose variables are the coordinates of h. The algorithm solves the resulting system of inequalities using an efficient linear programming algorithm (recall that this problem is in $\mathcal{P}$). Its output $\hat{h}_s$ is the hypothesis for the hidden hyperplane h^*.

Note that this algorithm can be adapted to behave as an on-line algorithm, like the previous one. Namely, starting with $h^0 = 0$, on every subsequent input pair x_t, b_t, output a hypothesis h^t, as follows:

- **Correct classification:** If $b_t = \text{sign}(\langle h^{t-1}, x_t \rangle)$, set $h^t = h^{t-1}$.

- **Incorrect classification:** If $b_t \neq \text{sign}(\langle h^{t-1}, x_t \rangle)$, let h^t be the output of the linear program for the system of inequalities derived from the first t examples.

It is worth noting at least one obvious extension of the problem of learning hyperplanes, which is handled by both algorithms easily. This extension turns out to be extremely useful and demonstrates the power of this problem when combined with simple reductions. Hyperplanes partition $\mathbb{R}^n$ by a linear equation. Instead, we can consider partitions of $\mathbb{R}^n$ defined by polynomial equations of higher degree, which gives a much richer class of identification problems. Observe, however, that there is a simple reduction from the latter to the former. Specifically, consider polynomials $p : \mathbb{R}^n \to \mathbb{R}$ of degree d. Let $m \leq n^{O(d)}$ denote the number of monomials in a polynomial of degree at most d. Given a point $x \in \mathbb{R}^n$, it can be mapped to a point $x' \in \mathbb{R}^m$ that evaluates x in every such monomial.[8] Then, if $p' \in \mathbb{R}^m$ is the list of coefficients of p, the value $p(x)$ is clearly given by the *linear* form $\langle p', x' \rangle$. Thus, labeled examples for linear inequalities $(x, \text{sign} \, p(x))$ can be converted to labeled examples for linear inequalities $(x', \text{sign}(\langle p', x' \rangle))$, and this data can be used by the algorithms above to identify p' and hence p. The cost of this reduction naturally increases with d. Various methods for achieving further efficiency and generalization come under the topics "support vector machines" and "kernel methods," which we will not discuss here.

---

[6]While similar, the focus is different. In on-line algorithms, the task is usually known in advance, and an algorithm designer can use arbitrarily sophisticated methods and analysis in solving it. In machine learning, usually there is far more limited information about what is to be achieved, and the eventual (prediction) algorithms extracted form the data are typically simple, coming from a small arsenal of methods.

[7]It is worth mentioning that the perceptron algorithm itself can be adapted to solve general linear programming problems [BD02].

[8]This is called the *Veronese map* (or embedding) in algebraic geometry.

17.2   Classification/Identification: Some choices and modeling issues

We will be using the terms *classification* and *identification* interchangeably, as different sets of literature use them for the same concept.

   The task at hand is to identify one, from a given collection of *functions* (sometimes called "concepts," or "predicates," or "rules"), using data that arrives in a stream of *labeled* examples. This task seems pretty concrete and focused. However, there are still important choices to make, and I list some of them. We will not discuss the *awareness* of the identification algorithm of these issues and the modeling decisions regarding them. This is a highly nontrivial component when discussing, for example, child (or animal) learning algorithms and how they evolved. But (as in previous chapters) it is completely reasonable to assume this awareness for algorithms that humans design, given such a specific classification task.

17.2.1   Target class of functions

This family of functions (often called *concept class*) is the scope from which we are trying to identify a single function (or concept). Typically it is a collection $F = \{f : X \to Y\}$ of functions from a fixed domain X to some fixed range Y, and the task is to identify one of them. The domain X and range Y can each be finite or infinite, even continuous (although for actual algorithms, continuous objects are typically represented by discrete approximations). Here are some examples. We will discuss some of these later.

1. All linear equations over a given field.

2. All polynomials over a given field.

3. All axis-parallel rectangles in the plane (e.g., representing all people in a certain age interval, and certain income interval). More generally, one can consider parallelepipeds in $\mathbb{R}^n$ (for properties with n attributes).[9]

4. All circles in the plane (e.g., representing all residences within a certain distance from some fire station or hospital). More generally, one can consider balls in $\mathbb{R}^n$.

5. All conjunctions of literals (variables or their negations), for example, $x_7 \wedge \neg x_2 \wedge x_4$ over a set of Boolean variables (e.g., representing the joint presence or absence of certain features like GPS, FWD, ABS, or cruise control, in the car you want to buy).

6. All DNF formulas (disjunction of conjunctions) over a given set of Boolean variables (e.g., representing your willingness to purchase a car with any one of a given set of features).

7. All functions computable by a finite automaton.

8. All functions in $\mathcal{P}$ (representing a property you can actually verify efficiently).

17.2.2   Hypothesis class

The identifying algorithm must respond to the given data with hypotheses. These come from a collection of functions $H = \{h : X \to Y\}$, which typically (but not always) contains F. As elements of H are the outputs of the learning algorithm, they are often specified by the class of algorithms (or machines) computing these hypotheses. For example, H may consist of small formulas, low-degree polynomials, finite automata, decision trees, Turing machines with resource limitations, and so forth.

---

[9]Here and in the next item, X is not discrete. However, we can replace the reals $\mathbb{R}$ with the rationals $\mathbb{Q}$ without any consequence for the issues at hand.

17.2.3   Admissible presentation of data

The way we have set things up, data arrive in a sequence of labeled examples $(x_i, f(x_i))$ for a sequence of points $x_i \in X$. A central issue is deciding how that sequence of points $\{x_i\}$ is chosen. Of course, to make learning models most general, one tries to assume the least about the way nature provides such examples in "natural" situations. This suggests letting an *adversary* generate the sequences.[10] Very broadly speaking, these adversaries can be limited in one of two possible ways (related to the two main evaluation criteria in Section 17.2.4 on quality measures):

1. The adversary can choose an arbitrary sequence, which eventually includes every point in X.

2. The adversary can pick a completely arbitrary probability distribution on X, after which the elements of the sequence are drawn independently from this distribution.

A crucial assumption we made above is that the labels of the data points are always correct. This of course in general is unrealistic, and a variety of relaxations have been studied, allowing both some fraction of noisy labels (random or adversarial) and perturbed labels (which are "small" in some metric over the range Y). We will not discuss these important generalizations here, although some of the positive results mentioned here extend to accommodate such errors, and many algorithms and heuristics are especially designed to tolerate them.

In an orthogonal direction, another natural form of data acquisition by learning algorithms allows them to ask *queries*. Numerous types of queries have been considered, including letting the algorithm choose some of the points x_i and requesting points x_i that violate the current hypothesis. Similarly, the number (or frequency) of such queries is another important parameter when they are allowed. Queries can add significant power to learning algorithms. We will not further consider such models here.

17.2.4   Quality measures for identification algorithms

What is a good identification algorithm? Ideally, one would want the algorithm to quickly learn the target concept (or function) so as to make few or no future mistakes in its hypotheses. Corresponding to the two general types of adversaries above, two notions of mistake bounds have been considered:

1. Completely stopping making errors after some finite number of samples.

2. Reducing the probability of errors as the number of samples grows.

The two represent very different philosophies with respect to learning in general—one that is more logically and linguistically oriented, and the other more statistically oriented. We will discuss them in Sections 17.3 and 17.4.

An important aspect for both approaches is the speed of learning. Two "input size" parameters are important to account for efficiency of identification algorithms. The first is the *length* of a single labeled example in the sequence; this is usually fixed, given the domain X and range Y (and often captures the dimensionality of the problem, e.g., the dimension n when identifying hyperplanes). The other is the *number* of examples (namely, the sample size) needed to obtain high-quality predictors of the target function. Ideally, the number of samples should be small, and the algorithm should be efficient in terms of both parameters. Perhaps surprisingly, there are important situations where there is a nontrivial trade-off between the number of samples and algorithm efficiency, which we will discuss later.

17.3   Identification in the limit: A linguistic/recursion theoretic approach

In a nutshell, this direction generally assumes that data arrive adversarially, allows some unspecified but finite "teaching" period, after which the "learner" has to get it perfectly right.

---

[10]As we will see at the very end of this chapter, adversaries considered in this chapter are too strong, in that very few concepts can be learned if they are unrestricted. And naturally, variants restricting them, as well as other relaxations have been studied as well.

The notion of *inductive inference* is almost as old as the theory of computation. One of the main original fields of study to drive it forward was linguistics, borrowing both from the computational perspective of computability and recursion theory, and from the scientific perspective of trying to understand how natural languages evolve and are learned (by humans and animals). An excellent survey of this direction of research is [AS83]. We only discuss some of its basic features and results.

The seminal paper, which has shaped this approach, was written by Gold [Gol67]. In this paper, he addresses the modeling issues considered above. In particular, he defines an important notion of success of an algorithm, namely *identification in the limit*. Gold also suggests a (simple) general technique that achieves such success, called *identification through enumeration*, and studies its power. Let me explain and give examples of both.

Fix a family of functions $F = \{f : X \to Y\}$. The admissible input presentation Gold considers is the first one we described above—an adversary selects a function $f \in F$, and then selects the order in which examples x_t appear, as long as every element of X appears at least once (and if X is finite, each element appears infinitely many times). An algorithm observes the sequence (x_t, b_t) with $b_t = f(x_t)$, and after every such example, it outputs a hypothesis $h_t : X \to Y$.

The class F is *identifiable in the limit* if there is an algorithm that, for every such adversary, makes only a *finite* number of mistakes. More precisely, after some finite time T, for all $t \geq T$, all h^t are the same (namely, $h^t = h^T$) and are correct (namely, $h^T(x_t) = f(x_t)$). We stress that the algorithm may not know what T is (namely, there is no requirement that the algorithm "knows" when it has stopped making mistakes). This is a clear (modeling) restriction of this learning model, which indeed makes it very strong and allows it to identify very complex function families.

We will now discuss three examples of targets, which have identification in the limit algorithms, to get a sense of what is learnable (and at what cost), and what is not.

Example 1: The class $\mathcal{P}$ Let $F = \mathcal{P}$, the class of Boolean functions on binary inputs computable by polynomial-time algorithms. A simple algorithm to identify this class in the limit applies the following simple idea, which Gold calls *identification through enumeration*. It uses a subroutine that enumerates all polynomial-time Turing machines (recall that each has a finite description, as does the integer exponent bounding the polynomial running time), namely, prints a list $M_1, M_2, \ldots$ of them all (some of them possibly computing the same function). Now, for every t, the identification algorithm selects the smallest n for which M_n in the list above is consistent with all examples seen so far (namely, for all $s \leq t$, $M_n(x_s) = b_s$), and outputs that M_n as its hypothesis h_t. To see that this algorithm identifies $\mathcal{P}$ in the limit, consider an arbitrary (hidden) $f^* \in \mathcal{P}$ used to label the data, and let k be the smallest integer for which M_k in the enumeration above computes f^*. It is clear that each of the M_n with $n \leq k$, if chosen as the hypothesis by the algorithm, will make a mistake after a finite number of examples. Furthermore, once M_k is chosen once, it will be chosen forever.

It should be clear that this algorithmic technique is very general. All it requires is two properties from the function class F. First, that there is an algorithm to enumerate F. Second, that each function in F is computable (to check consistency with the data so far). Gold observes that these properties hold in particular for all language[11] classes in Chomsky's hierarchy (finite, regular, context free, context sensitive, and recursively enumerable), and so all are identifiable in the limit.

So, identification by enumeration is extremely powerful, but it should also be clear that the identification algorithm underlying it can be arbitrarily inefficient. Moreover, even though all functions in the class $\mathcal{P}$ are efficiently computable and so the second property is efficiently testable, one may have to enumerate at step t an exponential (in t) number of machines before finding a consistent one (it is left as an exercise for the reader to check that it is never worse than that—the algorithm above will never run in time worse than exponential in the length of the data available). Of course, the running time for other target classes may be much larger. Our next example shows that in some cases, *efficient* learning in the limit can be achieved.

---

[11] Recall that a language (a set of sequences) is naturally associated to a function $f : \Sigma^* \to \{0, 1\}$ is the set of sequences f maps to 1.

Example 2: Rational polynomials   Let $X = \mathbb{Q}$, the field of rational numbers, and $F = \{p : \mathbb{Q} \to \mathbb{Q}\}$ be the set of all univariate polynomials. It is clear that the enumeration algorithm above applies as well to this target class, as it is enumerable and every function in it can be efficiently evaluated. Still, in an obvious implementation as above it will require exponential time, even for this simple subclass of $\mathcal{P}$. However, due to *interpolation*, one can do much better, as there is no need to search for the minimal consistent hypothesis—a unique one is well defined, and moreover, one can find it efficiently. To be more precise, here is how the identification algorithm works:

Start with some null hypothesis (e.g., $h^0 = 0$). On every subsequent input pair (x_t, b_t), output a hypothesis h^t, as follows:

- **Correct classification:** If $b_t = h^{t-1}(x_t)$, set $h^t = h^{t-1}$.

- **Incorrect classification:** If $b_t \neq h^{t-1}(x_t)$, let h^t be the unique degree $t - 1$ polynomial interpolating the first t examples.

The reader is invited to verify that if the hidden polynomial p^* used to generate the data has degree d, then by step $T = d + 1$, all hypotheses will be the same polynomial p^*, and the algorithm will run in polynomial time in the data length. The same naturally holds for polynomials over finite fields.[12]

Examples for which such an efficient identification in the limit is possible are rare. We will now see that it can be done as well for the first motivating target class we considered, namely, hyperplanes.

Example 3: Real hyperplanes   Recall from Section 17.1 the problem of identifying a hyperplane h^* from a sequence of examples. We gave two algorithms for the problem, the perceptron algorithm and linear programming. As it turns out, both efficiently achieve identification in the limit, in a somewhat weaker sense than Example 1 on polynomials; they converge to the correct answer after a finite number of mistakes, but this finite number depends on a parameter called *margin*, common to many identification (and more generally, learning) algorithms in continuous spaces. I will define it formally below, but intuitively, it captures the robustness of the data to small fluctuations. For example, in the spam mail example used to motivate hyperplane classification, one expects that for any pair of e-mails, one legitimate and one spam, there will be a significant, noticeable distance between them, and that the larger this margin is, the more efficient the classifier will be. We will now see this intuition in action.

We will only analyze the perceptron algorithm for classifying hyperplanes. Recall the algorithm again for convenience.

Start by initializing $h_0 = 0$. After the tth sample, set h^t as follows:

- **Correct classification:** If $b_t = \text{sign}(\langle h^{t-1}, x_t \rangle)$, set $h^t = h^{t-1}$.

- **Incorrect classification:** If $b_t \neq \text{sign}(\langle h^{t-1}, x_t \rangle)$, set $h^t = h^{t-1} + b_t x_t / \|x_t\|$.

To analyze this algorithm, define $\hat{x} = x/\|x\|$ to be the scaling (to a unit length) of a vector $x \in \mathbb{R}^n$. Also, introduce a parameter μ, called the *margin*, which is data dependent and measures the minimum distance of the points $\hat{x}_i$ to the hyperplane h^*. Thus, $\mu = \inf_i \langle h^*, x_i \rangle$. Note that the larger μ is, the better separation we have between the sets of points on the two sides of the hyperplane. Indeed, they are not separated by h^* but actually by a strip in the same direction whose width is 2μ. The margin determines a finite bound (which is independent of the dimension n) on the total number of prediction mistakes made by the perceptron algorithm.

Theorem 17.1 [Nov62]. *The total number of incorrect classifications of the perceptron algorithm on a data sequence of margin μ is at most $1/\mu^2$.*

An interesting observation is that even if the number of samples is finite (namely, the sequence $\{x_t\}$ contains only finitely many distinct points in $\mathbb{R}^n$), cycling through them sufficiently many times (depending on the margin) will lead the perceptron algorithm to find a hypothesis consistent with the data.

---

[12]The reader is invited to contemplate *multivariate* polynomials over the rationals or over finite fields.

Proof (sketch). The simple idea behind this proof of Novikoff has been used many times in analyzing other, more sophisticated algorithms; it rests on contrasting progress in the L_1 and L_2 norms. Intuitively, the correction made to the hypothesis h^{t-1} after every incorrect classification improves the correlation of h^t and the true separating hyperplane h^*. However, h^t is not much longer than than h^{t-1}, because the angle between h^{t-1} and x_t is obtuse, and $\hat{x}_t$ is a unit vector. Combining both facts will bound the number of mistakes. Let's make this idea formal.

We will bound, from above and below, the inner product $C_t = \langle h^t, h^* \rangle$, which is 0 at $t = 0$. Assume that the prediction of h^{t-1} on x_t is mistaken. For the lower bound, note that

$$\langle h^t, h^* \rangle = \langle (h^{t-1} + b_t \hat{x}_t), h^* \rangle \geq \langle h^{t-1}, h^* \rangle + \mu,$$

and so after N classification errors, $C_t \geq \mu N$. However,

$$(\langle h^t, h^* \rangle)^2 \leq \|h^t\|^2 \leq \|h^{t-1}\|^2 + 1,$$

where we have used the Cauchy-Schwarz inequality and the unit lengths of h^* and $\hat{x}_t$, along with the fact that there is an obtuse angle between h^{t-1} and $\hat{x}_t$. Thus after N errors, we have $C_t \leq \sqrt{N}$. Combining the bounds, we conclude that $N \leq 1/\mu^2$. □

17.4   Probably, approximately correct (PAC) learning: A statistical approach

In a nutshell, this direction assumes that data is generated randomly, insists on quantitative bounds on the number of labeled examples and on algorithmic efficiency, but allows unlimited prediction errors as long as they occur with low probability.

The notion of *distribution-free* learning was born in the seminal works of Vapnik and Chervonenkis [VC15, VC74], arising from the fields of statistical learning theory and probability theory, with a strong focus on *sample complexity*. A comprehensive treatment of this work and its origins and applications appears in Vapnik's books [Vap98, Vap13]. Independently but quite a bit later, Valiant [Val84b] came up with the same notion. However, motivated by understanding learning as a cognitive process, and the need to distinguish feasible and infeasible cognitive tasks in any theory of learning, Valiant made the *computational efficiency* aspect of learning algorithms central to his model. The term *probably, approximately correct* learning (or *PAC* learning for short) for this model was coined in [AL88]. An excellent intuitive introduction to this viewpoint on learning is the book by Kearns and Vazirani [KV94b].[13]

There are many essential differences between the approach of inductive inference taken in Section 17.3 and the statistical approach taken here. The main contrast is evident in the opening single-sentence summaries at the start of Sections 17.3 and 17.4. The logical framework of inductive inference is unforgiving to prediction errors. To achieve such perfection eventually, it is literally willing to expend an arbitrarily long teaching phase. In contrast, the statistical framework forgives prediction errors if they are rare, and insists on a short teaching phase and efficient learning. While mathematically both are very interesting, considering theoretical and practical work on computational learning today, one can say with certainty that the statistical approach has won big time. This is true even for the original motivation of the logical approach, namely, understanding the evolution and learning of languages, and translating and generating linguistic text. Perhaps the strongest reason for this advantage of the statistical approach is that in nature, *inefficient* learning can be far more damaging (for surviving and thriving) than *imperfect* learning. Moreover, essentially all practical products of machine learning are based on this view. Finally, the statistical, complexity-theoretic view also seems to suggest better models explaining how natural mechanisms of learning may have evolved; this view is elaborated in Valiant's book [Val13]. We return to the concrete task of classification in this framework.

---

[13]Note that in the literature, "PAC learning" is sometimes taken to mean the original distribution-free learning of Vapnik and Chervonenkis (which disregards computational complexity), and sometimes as the *efficient* version of Valiant. I will stress the efficiency aspect of learning algorithms in the definitions and results below, distinguishing "PAC learning" and "efficient PAC learning."

17.4.1   Basics of the PAC framework

For simplicity, we restrict ourselves from now on to identification of Boolean functions, namely, the range $Y = \{0, 1\}$. This case is often called *pattern matching* in the statistical learning theory literature and *binary classification* in the machine learning community. Extensions of the theory were studied for larger ranges Y, which can be either discrete or continuous (e.g., see the books cited above). Unlike the Boolean domain, where for every data point in X a hypothesis is either correct or incorrect, the quality of a hypothesis on a point (namely, how much it disagrees with the correct answer) has to be defined, often using some metric on Y, or more generally, a *loss function*. The study of this general setting is often named *empirical risk minimization*, where the "risk" is taken with respect to the loss function.

Back to Boolean ranges. Fix a target class of functions $F = \{f : X \to \{0, 1\}\}$, and furthermore an arbitrary probability distribution[14] D on X. In distribution-free PAC learning, admissible data to a classification algorithm (sometimes called *classifier*) is a sequence of labeled examples $(x_t, f^*(x_t))$, where the samples $x_t \in X$ are chosen *independently* according to D, and f^* is the hidden function we are trying to identify (or *classify*). Again, an algorithm produces a sequence of hypotheses h^t after observing the first t examples.

Let me first explain intuitively what a good algorithm in this setting is, and then define it more precisely. An algorithm is good if after some T steps, the hypothesis it outputs predicts the value of f^* on the next point with high probability. We stress that the number of necessary examples T does *not* depend on the underlying distribution D (this explains the term *distribution-free*). This sample size T can and will of course depend on properties of the target concept class F and on the two error parameters (accuracy and confidence, defined below) that govern the quality of prediction after so many examples.

There are two issues to discuss regarding the distribution D. First, it is stationary and does not change throughout the process; if this D is viewed as an environment generating experiences for the learner, then we can view stationarity as fairness: The learner is tested on experiences of the same type it has learned from.[15] This assumption is sometimes called *invariance*, and it is certainly a strong one. Even stronger is the assumption of *independence* of the samples. Both assumptions ignore (or sweep under the rug) the fact that in nature and practice, the hypothesis of the learner often generates action/behavior that affects the environment and future examples it may generate.[16] Still, these assumptions are natural in an initial mathematical model, and moreover, we will see that only a few general classes are learnable even with these assumptions. Once we accept them, every sample is as good as any other, and so from now on we'll consider a *training set* (rather than a sequence) of T labeled examples, and a hypothesis h is tested on a single random sample from D. With this, we are ready for the formal definition of *PAC* learning.[17]

Definition 17.2 [VC15, Val84b]. A concept class $F = \{f : X \to \{0, 1\}\}$ is *PAC*-learnable if there is a, possibly probabilistic, (learning) algorithm A and an integer-valued function $T = T(F, \epsilon, \delta)$ with the following property. For every probability distribution D on X, and for every function $f^* \in F$, on inputs $\epsilon, \delta > 0$ (respectively, the *accuracy* and *confidence* parameters) and $t \geq T$ independent examples from D labeled by f^*, the algorithm A returns a hypothesis $h = h^t$ that with probability at least $1 - \delta$ satisfies $D(h \triangle f^*) \leq \epsilon$. Here $h \triangle f^*$ is the subset of X on which f^* and h disagree, and $D(h \triangle f^*)$ denotes its mass under D. The success probability is computed over distribution D generating the examples, and any coin tosses of the algorithm A.

The algorithm A is called *efficient* if it runs in time polynomial in the parameters $1/\epsilon, 1/\delta$, and the total length of the T samples.[18]

If the algorithm A is restricted to output hypotheses only from the target class F, then this class is called *proper* PAC-learnable.

---

[14]Or measure, in continuous domains X; we will not be concerned here with measurability issues, which are not central to this topic.

[15]As in high school, when students demand that the test contain only questions previously discussed in class. Or, as in the wild, when African lions test their hunting strategies near the same water supply antelopes come to drink at every evening.

[16]Note that this objection does not affect the previous learning notion of "identification in the limit."

[17]I prefer *identification* to *learning*, but this notion is very common in the literature.

[18]It would stand to reason to also demand that T itself is small in terms of the error parameters, and properties of F. As we shall see, this is guaranteed automatically.

Let us relate the sources of the words "probably" and "approximately" in the PAC acronym to the two error parameters. *Probably* is associated with δ—the algorithm must produce a good hypothesis with high probability: at least $1 - \delta$. A good hypothesis is one that is *approximately* correct: The probability that h and f^* disagree on a random sample from D is at most ϵ.

Which classes of functions are PAC-learnable? And if so, by which algorithms? These two general questions have very clean answers, that are motivated and explained below.

Remarkably, a simple single combinatorial parameter of the target concept class F determines whether it is PAC-learnable or not. It is called the *VC*-dimension, after its inventors Vapnik and Chevonenkis [VC15]. Intuitively, it captures how "rich" the class of functions F is when restricted to any finite set of elements of the domain X. For a finite set $S \subset X$, let F_S denote the set of all restrictions of functions in F to S. That richness of F is the size $|F_S|$ as a function of $|S|$ for the worst possible S. This richness happens to be determined by the largest S for which F_S is maximal, namely, S is fully "shattered" by F. Formally, we call a set $S \subset X$ *shattered* if $|F_S| = 2^{|S|}$, that is, every possible Boolean function on S can be extended to a function in F.

Definition 17.3. The VC dimension of F, denoted $\text{VC dim}(F)$ is the largest size of a shattered set in X. If no such largest set exists, define $\text{VC dim}(F) = \infty$.

Try proving the following VC dimension bounds on some of the function classes we discussed. Doing so will clarify the definition and demonstrate that even when F is large, infinite, or even uncountable, it can have small VC dimension.

1. The VC dimension of axis-parallel rectangles in the plane is 4. More generally, the VC dimension of all axis-parallel boxes in $\mathbb{R}^n$ is $2n$.

2. The VC dimension of hyperplanes in $\mathbb{R}^n$ is $n + 1$. Moreover, the VC dimension of any collection of hyperplanes with margin (see example 3 in Section 17.3) at least μ (in any dimension) is at most $1/\mu^2$.

3. The VC dimension of conjunctions over n Boolean literals is n.

4. The VC dimension of any finite family F is at most $\log|F|$. In particular, the VC dimension of the class of all Boolean functions having a size-s DNF formula, (or even having a size-s Boolean circuit) is at most s^2.

The punchline is that this simple combinatorial parameter, the VC dimension of function classes, determines PAC-learnability and yields an optimal learning algorithm with optimal learning rate (namely, the necessary sample size for given error parameters).

Theorem 17.4 [VC15]. *A class F is PAC-learnable if and only if $\text{VC dim}(F)$ is finite. Moreover, denoting $\text{VC dim}(F) = d$, the following conditions hold.*

- *The number of required examples is $T(F, \epsilon, \delta) \leq O(\frac{1}{\epsilon}(d \log \frac{1}{\epsilon} + \log \frac{1}{\delta}))$.*

- *Every algorithm that produces as a hypothesis any function in F that is consistent with the sample, achieves the required error bounds.*

Let me stress a few points. First, the sample size bound is *independent* of the sizes of the domain X or the target class of functions F. In a precise sense, the VC dimension captures the "essential size" for the purpose of sampling from *any* distribution on X. Note that for constant error parameters ϵ, δ, the bound on the sample size is *linear* in the VC dimension of F. Finally, this upper bound on the sample size T is the best possible in all parameters, as proved in [KPW92].

This theorem has implications and interpretations beyond learning (and statistics, where it originated) to other areas, including discrete geometry (see [HW87]), discrepancy theory, and combinatorics, especially the study of set systems (hypergraphs). The connections to them all, as well as the fundamental nature of the theorem, arise from a somewhat more abstract version of it using the language of ϵ-nets (see, e.g., [Mat02, Chapter 10] for an exposition). For this view, it is best to think of functions in F as indicator functions of

subsets of X. For a distribution D, an ϵ-net is a subset of points in X that intersects every "large" set in F, namely, one whose D-measure is at least ϵ. The theorem says that, for any F of finite VC dimension, sampling enough points from *any* distribution D will be an ϵ-net for D with high probability. Moreover, a similar statement holds if one requires a stronger notion, ϵ-approximation (namely, a set whose intersection size with *any* set in F is within ϵ of its D-measure of that set). Using these connections, this theorem and its variants give *uniform* concentration bounds for potentially infinite (or even continuous) spaces X and function families F of finite VC dimension.

Now for a few words about the proof of the theorem. The intuition for the proof comes from the fact that a finite VC dimension d allows us to think of F as small, even when infinite. First let's see why a small sample suffices when F is small, and then see in what sense the VC dimension captures this smallness. Assume for simplicity that the failure probability δ is fixed to be some small constant, say, .001.

Our task is to show that for every $f^* \in F$, if we draw at random t points from D with t sufficiently large, any function $f \in F$ consistent with f^* on that sample will agree with f^*, with probability $\geq 1 - \delta$, on all but an ϵ measure (under D) of X. Consider any f for which $D(f \triangle f^*) \geq \epsilon$. Clearly the probability that the sample will miss that symmetric difference (and fail to distinguish the two) decays exponentially in ϵt, and so if $t \gg \frac{1}{\epsilon}$, this event will happen with probability at most δ. To prove the theorem for a finite F, we could do a union bound over all possible $f^*, f \in F$ and obtain the same failure probability by taking $t \gg \frac{\log |F|}{\epsilon}$.

The main point of Vapnik and Chervonenkis' proof is that $\mathrm{VC\,dim}(F)$ can (roughly) replace $\log |F|$ in the argument above, even when F is infinite. The reason is that if $\mathrm{VC\,dim}(F) = d$, on every finite sample size (say, t), then the functions in F can label these t points in at most t^d different ways (compare to the trivial bound of 2^t).[19] This seems to suffice for a union bound, as we have effectively reduced the number of pairs of functions to $t^{O(d)}$, which would be dwarfed by the $\exp(-\epsilon t)$ decay of failure probability to catch any single large symmetric difference. However, this idea does not quite work, as there is a nagging dependence between the functions at hand and the chosen sample. This subtle issue is handled with a slick argument that the reader is encouraged to discover or read about.[20] Another challenge I leave you with is proving this most basic fact about VC dimension used above.[21]

Lemma 17.5 [Sau72, She72]. *For any integers $d \leq t$, let $F = \{f : X \to \{0,1\}\}$ be any family of functions with $|X| = t$ and $\mathrm{VC\,dim}(F) = d$. Then*

$$|F| \leq \binom{t}{0} + \binom{t}{1} + \binom{t}{2} + \cdots + \binom{t}{d}.$$

17.4.2   Efficiency and optimization

While the VC dimension completely determines PAC learnability *in principle*, it is of course critical to have *efficient* learning algorithms, as Valiant insisted in his original definition [Val84b], and because these are the only ones we can hope to implement. We seek an efficient algorithm in terms of the VC dimension and the input and output size parameters (representation of an element of X, and a function from F).

Now the Vapnik-Chervonenkis theorem (Theorem 17.4) tells us that any hypothesis consistent with enough samples achieves PAC learnability. Let us check this for some concrete target classes. Going over the examples in the list in Section 17.4.1, it is simple to see that for the first three classes, efficient algorithms to find a consistent hypothesis exist.[22] On the other hand, it is not hard to prove that deciding whether a set

---

[19]This ingredient of the [VC15] proof was independently discovered in combinatorics by Sauer [Sau72] and in logic by Shelah [She72]; I state this simple and useful combinatorial fact more precisely below.

[20]Insufficient hint: Partition the random sample into two random parts of equal size to create the required independence.

[21]Insufficient hint: Use the following two steps. The easy one is proving the bound *if F is "downward closed"*: every function whose support is contained in that of a function in F is also in F. The harder one is reducing to this case by iterated "shifting": simultaneously remove some fixed $x \in X$ from the support of every function in F for which this removal defines a function outside F.

[22]For example 1, one can compute the minimal enclosing box of all positive examples by finding the minimum and maximum value in each coordinate. For example 2, one can use linear programming. For example 3, note that every positive example points to n of the $2n$ literals which cannot be part of the hidden conjunction, and can thus be eliminated. Repeating for all positive examples, the hypothesis which is the conjunction of all remaining literals will be consistent with the sample.

of positive and negative examples of a Boolean function have a size-s DNF, or a size-s Boolean circuit, are both $\mathcal{NP}$-hard problems. So this obvious approach to efficient learning algorithms does not work in these cases.

Note how convenient it is that PAC learning is equivalent to (the much simpler to understand) search for a consistent hypothesis with the data. This result naturally frames learning tasks as optimization problems and adds to the many connections between these two fields. And as usual in complexity theory, tasks of finding *approximately* consistent hypotheses and other relaxations are part of this study as well.

17.4.3 Agnostic PAC learning

One obvious shortcoming of the framework being considered is the assumption that the target class F is known (namely, that we have substantial structural information about the phenomenon we are observing and trying to classify). This may be true in some cases but false in many others. What happens if we drop this assumption completely? Note that, in contrast to F, the hypothesis class H (namely, the set of models we choose as possible explanations of the phenomena observed) is certainly within our control in science, and perhaps is easier to define or bound even when considering the learning abilities of living systems.

An important extension of Theorem 17.4 is to a model called *agnostic learning*, in which only H is present. It is interesting and important in the many cases as above, when the target class F is unknown, or when the hypothesis class H does not necessarily contain F. What can be the goal of the learner in this model? Before, we knew that some function f^* in F labeled the data, and the learner's goal was to approximate it with some hypothesis $h \in H$. Now f^* may be arbitrary. However, as only hypotheses in H are allowed, clearly the best one can hope for is doing as well as the best function $h^* \in H$ with respect to the data. How close can a learning algorithm come to that performance?

As it turns out, the very same proof of Theorem 17.4 above shows that the VC dimension of the hypothesis class H plays the same role, and the same algorithm achieves the same performance in this general setting. That is, any hypothesis $h \in H$ consistent with the number of examples in this theorem will be ϵ-close to the best one, h^*, under any distribution D, with probability at least $1 - \delta$. Indeed, it is this more general theorem that was proved in the original work of Vapnik and Chervonenkis (although the term "agnostic learning" came later).

17.4.4 Compression and Occam's razor

Some cognitive theories suggest that learning involves summarizing our life experiences in useful ways, where the emphasis is on *summary*, in contrast with just memorizing all of them. Similarly, scientists, after amassing data about some phenomenon, attempt to summarize it into a much more compact scientific theory. Let us explore this possible connection between compression and learning, and possible formalizations of it in the framework of PAC learning.

A famous quote of William of Occam, a fourteenth-century English philosopher, is *"entities should not be multiplied unnecessarily."* This became a basic principle of learning, and more generally of scientific discovery, which is called *Occam's razor*. Today, we interpret the quote as meaning *"among equally accurate explanations of the same phenomena, always prefer the simplest one."*[23] Although this principle sounds reasonable, it deserves an argument explaining the benefits of simpler explanations. As it turns out, PAC learning provides a framework in which Occam's razor can be *proved*, in the sense that using it implies learning. This was suggested by Blumer et al. in [BEHW90], which I now explain.

Consider only proper learning ($H = F$). The notion of simplicity we will use is the *description length* (in some canonical encoding) of functions f in F, denoted by $s(f)$.[24] The following theorem states (roughly) that if an algorithm can always compress a large number of labeled examples to a *shorter* consistent hypothesis, than the same algorithm PAC learns F.

---

[23]A well-known response to the natural question of why Occam's somewhat obscure statement was interpreted this way is that, well ... it is the simplest possible explanation. Luckily, many other thinkers, some as early as Aristotle, articulated this scientific principle more clearly much earlier.

[24]For example, there are simple binary encodings of various classes of circuits, formulas, finite automata, polynomials, geometric shapes, and so forth.

Theorem 17.6 [BEHW90]. *Assume that for some $\alpha < 1$ and $C > 0$, there is a (compression) algorithm A with the following property: on any sample of m inputs labeled by any function $f^* \in F$, with $m \geq s(f^*)^C$, A produces a function $f \in F$ that is consistent with the sample and $s(f) \leq m^{1-\alpha}$. Then A is a PAC learning algorithm for F.*

The proof is simple. It uses the exponential (in m) decay of probability of missing a significant (under the given distribution) part of the space, together with a union bound on the possible number of functions with such a small description. Intuitively, this says that a simpler consistent explanation should be preferred, because it is more likely to be flagged as incorrect if it indeed is incorrect.

This formal one-directional connection between compression and learning raises the question of a converse direction: Does learnability imply compression in some formulation of these concepts? In their paper [LW86], Littlestone and Warmuth define a parameter of arbitrary classes F, which we call here $Comp(F)$, capturing the compressibility of large labeled samples from F. Roughly speaking, $Comp(F) \leq k$ if any sample (of any size m) of labeled examples can be compressed to a subsample of only k labeled examples plus an additional k bits of size information, in a way that allows complete recovery of the labels of the $m - k$ "deleted" examples.

Their main result is that here, too, compression implies learnability, as it simply bounds the VC dimension: VC $\dim(F) \leq O(Comp(F))$. They asked if in one can prove any relation in the other direction. This took 30 years to resolve; Moran and Yehudayoff [MY16] proved $Comp(F) \leq \exp(\text{VC} \dim(F))$. Thus, the finiteness of one parameter implies the finiteness of the other, and compression characterizes learnability in this framework! The next theorem summarizes these results.

Theorem 17.7 [LW86, MY16]. *There is a universal constant $c > 0$ such that the following holds. Let F be any class of functions with VC $\dim(F) = d$ and $Comp(F) = k$. Then*

$$k/c \leq d \leq \exp(ck).$$

It remains a very interesting problem to close that exponential gap.

17.4.5   Boosting: Making weak learners strong

The idea of *boosting* is one of the central ideas in machine learning, applicable beyond the PAC model where it was born. Roughly speaking, it is a way of creating highly accurate predictors from rules that just barely do better than guessing. The idea sounds preposterous, especially if you consider the possibility of using your favorite stockbroker or weatherman, who perhaps can guess some bit (appropriate to their expertise) about tomorrow with (say) 51% probability, and turning their advice over time into a method that will let you succeed 99% of the time in the same prediction task. However, in quite general settings, this *is* possible. This was first proved in the PAC model in the breakthrough paper[25] of Schapire [Sch90], and soon after was simplified and improved by Freund [Fre90]. Boosting then rapidly evolved beyond this model and computational learning, and was applied to optimization, statistics, game theory and beyond. (Some of these applications were noted already in the original paper, including implications to the space efficiency of learning algorithms, as well as the first $\mathcal{NP}$-hardness result of learning tasks.) We have already seen incarnations of boosting in two different contexts earlier in this book: on-line expert prediction in Section 16.2 and structure versus randomness in Section 8.8. Boosting is part of numerous practical systems and is useful in scientific experiments. This theory of boosting and its scientific and practical applications are explored in the excellent book [SF12] by the originators. Here we restrict ourselves to the PAC framework.

We will now fix a target class F and a hypothesis class H. Let us return to the definition of PAC learning. The quality of a PAC learning algorithm is governed by two parameters: ϵ, the *accuracy* and δ, the *confidence*. We wish to study the potential of improving the quality of these parameters.

Making explicit their roles in the definition of PAC learning, let us call an (efficient) algorithm A an (ϵ, δ)-learner if for every distribution D, on sufficiently many independent examples labeled by some $f^* \in F$ returns, with probability $\geq 1 - \delta$, an ϵ-accurate hypothesis $h \in H$ (namely, one satisfying that $f^* \triangle h$ has measure at most ϵ under D).

---

[25]Yet again, by a graduate student!

We define F to be PAC learnable using hypotheses from H if such an (ϵ, δ)-learner exists for *every* choice of $\epsilon > 0$ and $\delta > 0$. (Indeed, in the standard definition, ϵ and δ are given as inputs to the learning algorithm as required accuracy and confidence parameters, respectively). However, suppose that someone provides us with a (ϵ_0, δ_0)-learner A for some fixed ϵ_0, δ_0. Is there a way to use A to construct (efficient) learners with better parameters? If so, under what conditions? The answers are *yes* and *always*; learning under the weakest possible parameters implies learning with the strongest ones.

Theorem 17.8 [Sch90]. *For every $\epsilon_0 < \frac{1}{2}$ and $\delta_0 < 1$, an efficient (ϵ_0, δ_0)-learner for F with H implies efficient PAC learning of F with hypotheses H', where functions in H' are efficiently computable when those in H are.*

The remainder of this subsection is devoted to sketching the proof of this important theorem. We will now argue its two parts: first, the simple amplification of *confidence*, and second, the deep and fundamentally important amplification (namely, boosting) of *accuracy*. For both parts, we will focus only on these parameters, mostly ignoring effects on the cost in sample size and computational efficiency (for the simple reason that controlling these effects is rather routine). We remark that these costs will only degrade polynomially in the gaps of the important parameters from the trivial bounds, namely $\frac{1}{2} - \epsilon_0$ and $1 - \delta_0$.

Let us treat one parameter at a time. Attaining any desired confidence $\delta > 0$ is easy to achieve, starting from any $\delta_0 < 1$. The main idea is that labeled data can be used not only to train classifiers, but also to test their quality. So, consider running A k times, on independent samples, with $k = O((\log 1/\delta)/\delta_0)$. This produces k hypotheses, $h_1, h_2, \ldots, h_k$. Now pick another (large enough) independent sample S, and test the quality of each h_i on S. Finally, output h_i for $i \in [k]$ which made the fewest errors on S. It is not hard to see that this new algorithm is an (efficient) (ϵ_0, δ)-learner with the confidence parameter δ specified.[26]

How about improving the accuracy parameter? Note that this is far less trivial. To see this clearly, let us consider a "toy case" in which the given learner A has confidence 1 (equivalently $\delta = 0$).[27] Namely, on sufficiently many examples labeled by any $f^* \in F$, drawn from an arbitrary distribution D, the given learner A always returns a hypothesis h that disagrees with f^* on at most ϵ_0 measure of D. How on earth can we produce a hypothesis which makes fewer errors than h?

The answer lies in making full use of the PAC model. Recall that while we want to learn f^* under the distribution D, the learner A is guaranteed to successfully learn it under any other distribution D'. Schapire's ingenious idea is to actually provide A with labeled examples drawn from other (judiciously chosen) distributions, which will help identify points on which h errs and correct for them. You might now ask: How can we sample from other distributions? After all, the distribution D generating examples is not under our control. To which Schapire's answer would be, simply by *filtering* (or reweighing) examples arriving from D.[28] Let us see these ideas in action.

The most natural way to proceed here be to describe and analyze the algorithm AdaBoost of Freund and Schapire [FS95], possibly the most elegant and practically useful boosting algorithm.[29] However, AdaBoost is described well in so many sources; moreover, its analysis is very similar (to the closely related) on-line expert prediction algorithm described in Section 16.2. So instead of AdaBoost, I describe here the original boosting algorithm of Schapire [Sch90], which is less well known. While less efficient, its analysis is more intuitive. Moreover, it is similar to a very different amplification algorithm in circuit complexity, that of Valiant [Val84a] for constructing short monotone formulas for the Majority function.

I will show how to use A, which is an ϵ_0-learner,[30] to construct an ϵ_1-learner B, with $\epsilon_1 < \epsilon_0$. Iterating this construction will lead to a sequence of ϵ_i-learners, with the sequence $\{\epsilon_i\}$ quickly converging to 0. We

---

[26]A sentence starting with these words is often false, as is the case here. But morally it is correct. The slightly uglier correct statement is that for any $\gamma > 0$, the above construction can yield a $(\epsilon_0 + \gamma, \delta)$-learner. Moreover, the dependence of k and $|S|$ on $1/\gamma$ are polynomial, making the construction efficient.

[27]Actually, by the amplification of confidence above, this toy case is essentially as general as the general case, as we can drive δ as close to 0 as we wish. Assuming $\delta = 0$ will release us from accounting for the (minor) way in which this parameter deteriorates when amplifying accuracy.

[28]As possibly corrupt casinos might do when making sure that the roulette ball lands in a random slot except for those clients who bet on it in this round.

[29]The name AdaBoost comes from "adaptive boosting"—when amplifying the accuracy, it does not even need to know the initial accuracy ϵ_0 of the given learner A.

[30]As discussed, we disregard δ, as it is set to 0.

will not analyze this simple convergence and the sample size and efficiency analysis, and just proceed to analyze one amplification step.

The learner B will behave as follows. It will feed A independent samples from three distributions (specified below), D_1, D_2, D_3, on which A responds with outputs hypotheses h_1, h_2, h_3, respectively. Then B will output as hypothesis the function h, which computes the majority of the h_i's (namely, on every domain element $x \in X$, $h(x) = Maj(h_1(x), h_2(x), h_3(x))$).[31]

The distributions are specified as follows. $D_1 = D$, the original distribution. D_2 rebalances the probability of elements in D, so as to give equal weight $\frac{1}{2}$ to those for which h_1 is correct and those for which it is wrong. Note that, as we have h_1 at hand, this reweighing is easy.[32] Finally, D_3 outputs only labeled examples from D for which h_1 and h_2 disagree (discarding all others). One can check that this sampling from D_2, D_3 can be done efficiently, given samples from D.

The main quantitative claim is that if the accuracy of A satisfies $\epsilon_0 \leq \alpha < \frac{1}{2}$, then the accuracy of B satisfies $\epsilon_1 \leq 3\alpha^2 - 2\alpha^3$, which is strictly smaller than α for $\alpha < \frac{1}{2}$. The observant reader might have noticed that this formula is familiar: when tossing a coin whose probability of heads is α three times independently, the probability that the outcome will have a majority of heads is exactly $3\alpha^2 - 2\alpha^3$. Now think of $h_i(x)$ when x is sampled from D as three coin tosses; it is clear that each of them mislabels x with probability at most α. However, they may be far from independent (depending on A's behavior). Despite this, the following simple argument shows that the worst (largest) probability that $h(x)$ mislabels x is when they are independent (namely, is at most $3\alpha^2 - 2\alpha^3$).

For this analysis, divide the domain X into 4 parts: $X_{CC}, X_{CW}, X_{WC}, X_{WW}$, where the first subscript encodes whether h_1 was correct or wrong, and the second encodes whether h_2 was correct or wrong. Denote the measures (under D) of these four sets by a, b, c, d, respectively. Now we use the promised accuracy of A on any distribution, in particular, D_1 and D_2.[33] As h_1 and h_2 are each wrong with probability at most α on their respective distributions, we get (please verify) the following two inequalities:

$$d + b \leq \alpha, \qquad\qquad c/(2(1-\alpha)) + d/(2\alpha) \leq \alpha.$$

Using these two inequalities, we can bound the probability that h errs on D as follows. For the majority to be wrong, either x falls in X_{WW} (which has measure d), or it lands in the part of $X_{CW} \cup X_{WC}$ for which h_3 was wrong (which is a fraction of at most α of that set). Thus, h is wrong with probability at most

$$d + \alpha(b+c) \ \leq \ [\alpha(d+b)] + [(1-\alpha)d + \alpha c] \ \leq \ \alpha^2 + 2(\alpha^2 - \alpha^3) \ \leq \ 3\alpha^2 - 2\alpha^3.$$

17.4.6   The hardness of PAC learning

So, which classes of Boolean functions can be PAC-learned efficiently (in terms of their description size)?[34] Valiant showed in his original paper that the class of conjunctions (of literals and their negations) can. This is nice, as often we classify objects (animals, plants, preferred cities, homes, friends, etc.) by the presence and absence of certain features (each of which can be represented by a Boolean variable). But obviously, this is a very primitive description of objects. Suppose we move one level up and ask for disjunctions of conjunctions, namely, DNF formulas? Surprisingly, this basic question, which has now been under attack for decades, is still open (and the general belief is that the answer is negative).[35]

Here we discuss how one argues the hardness of a PAC-learning task, and how one deals with such hardness. Let us tackle these questions one at a time.

---

[31]This clarifies that the final hypothesis class H' after iterations will be simple formulas of the functions in the original hypothesis class H.

[32]For instance, for each example, first flip a fair coin to choose "correct"/"wrong," and sample labeled examples from D, taking the first for which h_1 agrees/disagrees with the label.

[33]Note that in D_2 we reweigh every element on which h_1 was correct by $1/(2(1-\alpha))$, and every element on which h_1 was wrong by $1/(2\alpha)$.

[34]For natural classes of Boolean functions (e.g., various circuits, formulas, branching programs), their description size also bounds their computational complexity. Additionally, efficient learning can only be done for succinctly described functions.

[35]You may recall that in Section 17.4.2, we mentioned that computing a DNF formula of a given size consistent with a given sample is $\mathcal{NP}$-hard, but of course, this is only a sufficient condition for PAC learning.

The first to consider hardness results were Kearns and Valiant [KV94a], who made the following basic connection to cryptographic hardness. First, they noticed that random functions are hard to learn (even under the uniform distribution). After all, *by definition*, seeing the value of a random function on a sample of inputs tells you absolutely nothing about its value on any other input! Of course, this in itself is not such bad news, as random functions cannot be described succinctly. But now recall the cryptographic pseudo-random functions of [GGM86] (mentioned in the chapters on cryptography and randomness), constructed under the assumption that one-way functions exist (see Section 4.5). This is a family of efficiently computed functions (each has a polynomial-size circuit), but a random member of this family cannot be *efficiently* distinguished from a real random function. Thus, they cannot be learned, either! So, the first message is that *not everything that is efficiently computable is efficiently learnable.* More generally, this establishes the following principle: If a class of functions contains pseudo-random ones (under certain complexity-theoretic assumptions), then it cannot be PAC-learned, even under the uniform distribution. Kearns and Valiant [KV94a] give some other examples of such classes, and Kharitonov [Kha93] extends their results to very low complexity classes.

But short DNFs are extremely simple functions, so what does this approach tell us about their learnability? It is not hard to see that if we only care about learning under the uniform distribution, there is a simple algorithm that learns DNF formulas of size n in quasi-polynomial ($n^{O(\log n)}$) time.[36] In contrast, the best known PAC learning algorithm (for arbitrary distributions), due to Klivans and Serveidio [KS01], takes mildly exponential (roughly $\exp(O(n^{1/3}))$) time.[37]

The hardness of PAC learning k-DNFs was first considered by Pitt and Valiant [PV88], who showed that for *proper* PAC learning (namely, when the hypothesis class should itself be k-DNF formulas), the problem is $\mathcal{NP}$-hard (under randomized reductions). This was extended to the hardness of proper learning of DNFs by DNFs in [ABF+08].

But proper learning is an unreasonably strong demand of a learner, and indeed, the best known algorithms are not proper (both [KS01, Jac94] produce hypotheses that are polynomial threshold functions). Only recently, Daniely and Shalev-Shwartz [DSS14] finally proved general PAC learning hardness under a natural and reasonably well-studied average-case hardness assumption: that no fixed polynomial-time algorithm can *refute* random k-SAT formulas for growing k (see the paper for the precise assumption and the many results supporting it).

While we have focused on DNFs, many other very simple function classes are known or suspected to be hard in the PAC model (sometimes in the agnostic or noisy variants). Many other learning models (with added features like query access, or more forgiving error measures for non-Boolean labels) besides PAC exist and are realistic in many situations. In some such models, for some of these simple problems, we do have provably efficient learning algorithms. However these models are very far from explaining the empirical success of humans, animals, and many machine-learning heuristics in learning much more complex functions, even under unknown (and sometimes even varying) distributions. This dichotomy, the efficiency gap between various heuristics on "real life" instances on the one hand and algorithms with provable guarantees on the other, goes far beyond learning theory, to algorithms and optimization. Building models that better capture real-life inputs and problems remains a major challenge for computer science as a whole. But in machine learning especially, recent progress in training "deep networks" to learn highly complex structures seems to be pushing the field into accepting (certainly for working systems) successful heuristics, even when we have no idea what they are doing and how. The impact of this trend on the nature of the field, and on society, has yet to be seen!

It is thus good to conclude this discussion with a concrete mathematical open problem for which I believe there should be a provable result. It is a very special case of DNF learning, under the very special uniform distribution. It was suggested by Blum [BL97], and captures yet another basic problem of learning: the elimination of irrelevant features. Let F_k be the class of Boolean functions on n bits, which actually depend on at most k of their input bits. Can the relevant (and irrelevant) bits be efficiently detected from uniformly

---

[36]And in a somewhat more general model that allows the learner to ask queries, Jackson's celebrated Harmonic Sieve algorithm [Jac94] (which cleverly uses boosting) can learn DNFs under the uniform distribution in polynomial time.

[37]Via a clever reduction to learning hyperplanes in these many dimensions, by showing that DNFs of size n can always be represented as the sign of a real polynomial of degree roughly $O(n^{1/3})$. Incidentally, this degree bound is tight by a result of Minsky and Papert [MP69], an important early text in computational learning theory.

random labeled examples?[38] Note that for constant k, one can efficiently learn F_k by trying all possible subsets, and that for $k \leq \log n$, F_k is a subclass of linear size DNF formulas.

Open Problem 17.9. Find a polynomial-time algorithm for learning F_k under the uniform distribution for any k growing with n (or argue hardness using a natural complexity assumption).

An interesting trade-off is proved for this problem in [KRT17]: For $k = \log n$, any learner needs a super-polynomial number of samples, and hence super-polynomial time, assuming it has only linear size memory $O(n)$.

---

[38]Clearly, any learning algorithm will reveal this partition.

18 Cryptography: Modeling secrets and lies, knowledge and trust

Cryptography is a truly vast field. It serves as a foundation for its more practical sibling of computer security, which employs numerous computer professionals around the globe and is a major part of the computer industry. Cryptography is also directly connected to the very related but distinct field of digital privacy, always in the news for obvious societal concerns, and for which a beautiful new theoretical framework has recently been developed (see, e.g., the surveys [DR14, Vad17]). A comprehensive text on the modern foundations of cryptography is Goldreich's two volumes [Gol04].

We have discussed aspects of cryptography a few times in this book, as it provided important notions and directions of study in computational complexity. I presented the most basic assumptions of complexity-based cryptography, namely, one-way and trap-door functions, in Section 4.5. We discussed the cryptographic origins of pseudo-randomness in Section 7.3 and the concept of zero-knowledge proofs in Section 10.2.

In the following overview, I would like to focus on more general aspects, especially the complex mathematical modeling of cryptographic situations and some of its principles. In particular, I will highlight the axioms underlying modern cryptography; the variety of tasks and constraints, basic primitives, and design paradigms in modeling these; the key use of randomness and computational indistinguishability; and the fertile interaction between the complexity-theoretic and information-theoretic viewpoints, techniques, and results. After this hefty material, which is mainly classical, I will briefly describe three cryptographic tasks on which there has been exciting recent progress. After describing this elaborate and beautiful theory that underlies computer system security, we return to reality, and conclude with a discussion of physical, rather than mathematical, attacks on practical cryptographic systems.

18.1 The ambitions of modern cryptography

For millennia, *cryptography* (or *cryptology*), as a human activity, has focused on the sole purpose of the literal meaning (in Greek) of these words: *secret communication*.[1] The main ambition of the field was designing codes that would allow certain parties to exchange private messages, while preventing others from understanding them (and from the other end, the main ambition was the breaking of such codes).[2] Secret communication was (and is) a hugely important goal in plenty of contexts. We will not discuss the rich history of this ancient field; two comprehensive popular books which do are [Sin11, Kah96]. Instead, I'd like to point out that secret communication is but one (important) task that is needed to enable and protect privacy and integrity. The ambition of classical cryptography was focused in this *one* task.

But in numerous human activities, enabling and protecting privacy and integrity presents a challenge. You'd like no one to be able to read your secret diary (or files). You would like to withdraw cash from your bank account, but no one else should be able to. You'd like to cast your vote but should not be allowed to vote twice. You'd like to hold a private conversation with a friend that no one else should be able to understand. You'd like to place a bid without your competitor bidding $1 less 1 second later. You'd like to play poker without the other players peeking into your hand or keeping aces up their sleeves. The list goes on and on. The ambition of modern, complexity-based cryptography is to address *all* such tasks (and more).

Having to deal with all these complex situations in which the abovementioned tensions are present, humans have developed a vast array of physical means (typically task specific), like playing cards, sealed envelopes, supervised ballot stations, big locks on heavy safes, official bank notes, and ID cards. Modern cryptography allows no physical means whatsoever. We want to achieve everything as above, except that the "objects" possessed by individuals are information bits, and the actions are digital communications between computers (or, equivalently, verbal communications between people)!

Despite (or because of) these severe limitations, the seemingly impossible ambitions above were found to be possible through the theory of modern cryptography. The confidence to "ask for the moon" evolved for about a decade, from the late 1970s to the late 1980s, when successively varied and more difficult tasks

---

[1]The literal meaning is "secret writing" for cryptography and "the study of hiding secrets" for cryptology.
[2]Often called *cryptanalysis*, which in Greek means "uncovering the hidden."

were found possible in the new framework. It should be stressed that these developments happened *before* the Internet arrived on the scene. Moreover, this theory *enabled* e-commerce applications on the Internet, motivating its rapid expansion through private companies' investments! *Modern cryptography is one great example of the incredible economic consequences of curiosity-driven, theoretical research.*

I will soon lay out the complexity-theoretic axioms underlying modern cryptography, but first, let us contrast the information-theoretic and complexity-theoretic viewpoints.

18.2 Information theory vs. complexity theory: Take 1

Let us go back to the secret communication problem (which we shall return to again and again). All solutions invented for it through the ages had a common feature: that the two communicating parties secretly *shared* some piece of information (sometimes called a *key*), which the eavesdroppers did not have. In other words, it was always assumed that the communicating parties had (somehow) solved an *earlier* secret communication problem, namely, the secret communication of that key. This was an underlying axiom of classical cryptography, which we shall strive to eliminate.

Shannon, in one of the earliest applications of his just-developed "mathematical theory of communication" (which we today call "information theory"), sought to investigate precisely *quantitatively* the need for this axiom. His paper [Sha49a] formally defined the notion of a *perfectly secure* secret communication system, and proved that in any such system, secretly sending an n-bit message requires the pre-agreement on a perfectly random n-bit key. Namely, for each bit you want to communicate privately in the presence of eavesdroppers, you must pay with one bit communicated somehow without them present. So provably, this axiom cannot be eliminated!

Or can it? Probing deep into the proof of Shannon's theorem, one realizes that it relies on the following, completely reasonable (and seemingly benign) axiom of the information-theoretic view: *all representations of a random variable are equally useful.* More precisely, if X is a random variable on some set S, and $f : S \to T$ is any *bijection* from S to another set T, then for all purposes of information theory, the random variable $Y = f(X)$ is equivalent to X. To do what we have to do (namely, eliminate the need to privately communicate first in order to privately communicate later), we have to remove this axiom of information theory. Section 18.3 describes the axioms of modern, complexity-based cryptography, which form its foundation. But first, to illustrate the difference between the two viewpoints, consider the following example.

Fix a large number n, and define a probability distribution by picking, uniformly and independently, a pair of n-bit primes p and q. Consider the following two random variables X and Y defined by $X = (p, q)$, and $Y = p \cdot q$. Now, suppose I offer you one of them—which would you choose? Information-theoretically, these two random variables are equivalent: The value of one uniquely defines the value of the other (by unique factorization). So, it is two for the price of one, and you shouldn't care which you get. Unless … *time is of the essence* for you, and of course it is for everyone. From a computational perspective, there is a huge difference between the two choices. Y can be obtained from X *efficiently* via multiplication, whereas to obtain X from Y is equivalent to factoring, which is currently impossible to do efficiently. This *asymmetry* is the fundamental difference between the two viewpoints, and it is what allows numerous feats that are information-theoretically impossible to become possible in the complexity-theoretic model.

18.3 The axioms of modern, complexity-based cryptography

Modern, complexity-based cryptography, initiated in the seminal[3] paper of Diffie and Hellman [DH76], makes the impossible possible by introducing two complementary axioms:

- **Axiom 1:** All participants in cryptographic protocols are computationally bounded algorithms.

---

[3]The word "seminal" will appear many times in this chapter; no occurrence is taken lightly. This particular paper, besides the scientific and technological revolution it initiated, is an extremely lucid account of basic issues and ideas and their history—a must-read model of scientific writing.

- **Axiom 2:** There are computational problems that cannot be solved by such algorithms.

Before proceeding, let us recall that both information-theoretic cryptography and complexity-theoretic cryptography share a common axiom, which can be called Axiom 0: namely, that all participants have access to perfect randomness:[4]

- **Axiom 0:** All participants in cryptographic protocols can generate an unlimited number of independent, unbiased coin tosses.

Let us discuss these "axioms" before starting to use them. Of course, none of these axioms are as self-evident as some familiar mathematical axioms.[5] But a billion users of cryptographic systems that are based on these axioms implicitly put their trust in them—here not mere abstract truth is at stake, but rather your privacy or fortune.

Axiom 0 is at least well defined. It is the same one underlying probabilistic algorithms. While for efficiency of algorithms, it may not be essential (as discussed in Chapter 7), for cryptography, randomness is essential, as the very notion of a secret (and all other such notions) cannot even be defined without randomness. We have discussed randomness in earlier chapters, and in particular coping with situations when it is not perfect (as we assume). Note that in cryptography, this is a bigger worry than in algorithms (as it can impact security), but we shall not address this issue here. Assume Axiom 0 as is.

Axioms 1 and 2 require more elaboration. They are not precisely defined and indeed may be chosen to depend on the actual environment and application. For Axiom 1, this freedom is actually a source of modeling power; it allows putting different limitations on different resources (which may differ for different users) as the real-life situation at hand seems to demand. Axiom 2 should match Axiom 1, as the security of a system will rest on this intractability assumption. Before choosing which assumption to use, let us reflect first on why we actually need an intractability assumption at all. Sure, it would be great to have unconditional security proofs, not relying on any unproven intractability assumption. But for practically all cryptographic systems ever designed, breaking them is an $\mathcal{NP}$ problem,[6] and so proving their security will imply $\mathcal{P} \neq \mathcal{NP}$.[7]

So, we must make hardness assumptions, and a difficult question is which ones are acceptable. It would have been great to rest cryptography on $\mathcal{P} \neq \mathcal{NP}$, but we have no idea how to do this. Concrete suggestions like the hardness of factoring seem excellent, as this choice is mathematically elegant, has withstood centuries of attacks,[8] and is independent from the applications. But factoring may turn out to be an easy problem. This central question, of which assumptions are acceptable, is a subject of constant debate (especially given the wealth of assumptions this field has generated), and excellent papers suggesting concrete criteria for acceptability are [Nao03, GW11, GK16]. In this chapter, we focus on the most common choices. Namely, in Axiom 1, we limit all participants to polynomial-time computation, and we strive[9] to make Axiom 2 as general as the existence of trap-door functions[10] (see Section 4.5).

18.4 Cryptographic definitions

It is hard to convey the difficulty of precisely, mathematically defining cryptographic tasks and cryptographic protocols for them. Indeed, to get a real sense, one has to work in this field, or at least honestly delve into

---

[4]Relaxation of these axioms, when only "imperfect" randomness is available, was explored in cryptography and more generally in computation. We discussed modeling weak random sources and coping with them in Chapter 9.

[5]However, recall that the axiomatic foundations of mathematics itself, on which the "absolute truth" of theorems is based, have been and are still debated.

[6]Simply because one can guess the randomness and private inputs/keys of all participants.

[7]The optimism of the late 1970s is explicit in Diffie and Hellman's paper; they cite the recent advances in computational complexity, especially the discovery of $\mathcal{NP}$-completeness, as a central motivation for their complexity-based cryptography, and naturally hope that this field will soon produce intractability results!

[8]Gauss is on record for trying hard, but he certainly was not the first. As mentioned in the prelude (Chapter 2), even Euclid's GCD algorithm is the result of frustration with the difficulty of factoring when simplifying fractions.

[9]In many cases, initial security proofs for concrete systems rested on stronger assumptions, which were relaxed in later works.

[10]Which is of course implied by the hardness of factoring, but is far more general and thus may survive if factoring proves to be easy.

their details in different chapters of [Gol04], for example. Indeed, the advertisement above, that modern cryptography can achieve seemingly impossible goals, crucially depends on choosing the right definitions, which often are subtle and arrived at after subtle flaws were discovered in more naive, initial ones. I will try to give you an idea of some of the sources of these difficulties. First, unlike computation, which is typically performed by one algorithm that has the full problem input, here there are many interacting participants, each having part of the combined input. How do those algorithms communicate? By pairwise communication, or a broadcast mechanism? Can bad players eavesdrop? Corrupt messages? Collude in groups? All these decisions are crucial for accurate modeling of the desired application, as well as for the feasibility of a protocol. Second, unlike computational tasks, in which the desirable goal is almost always a correct answer, cryptographic tasks come with a huge variety of desiderata. Moreover, as some of the players may execute other algorithms than are required by the protocol, defining these properties requires nested alternations of quantifiers over the inappropriate behavior of bad players. I demonstrate these issues by a few examples of some cryptographic tasks. All of these were considered, formally defined, and proved possible in the 1980s, way before the Internet! It is an excellent exercise for you to try to come up with some semiformal definitions for their intuitively stated requirements. Feel free to use the axioms, and I stress again how they change the notion of perfect secrecy: What a player cannot *efficiently* compute from the information she has is perfectly hidden from her.

Note that in tasks with many participants, one has to specify the communication network between them. Throughout we assume a broadcast model, where everyone can hear what anyone says (imagine they are in the same room, or indeed, on the Internet); in this model, it is hardest to protect against information leaks. Here is a list of natural tasks.

- **Secret communication:** Alice and Bob, who may have never met before, want to exchange messages with perfect fidelity, so that Eve, who has full access to their communication, cannot extract *any* information about these messages from it. In other words, Alice and Bob should *publicly* create a *private* language.[11]

- **Message authentication/digital signature:** Alice and Bob, who may have never met before, want to exchange messages with perfect certainty of their origin, so that Eve, who can inject messages into their communication line, cannot impersonate either of them.

- **Secret exchange:** Alice has a secret S_A, and Bob has a secret S_B. A protocol must ensure a fair exchange: Alice learns S_B if and only if Bob learns S_A, and if they both follow the protocol, both do.[12] A similar "synchronization" issue arises in other tasks (e.g., contact signing).

- **Millionaires' problem:** Alice and Bob wish to find out who is richer, without any other information about their respective fortunes leaking to the other.

- **Zero-knowledge proofs:** Alice attempts to convince Bob that she has a proof of a mathematical statement known to both. If false, then Bob rejects their conversation, and if true, Bob accepts it but learns nothing from it beyond the truth of the given statement.

- **Collective coin flipping:** A set of mutually distrusting parties wishes to jointly toss and agree on a fair coin, which none of them can bias. A related problem is agreeing on a random "leader" among the set. Note that subsets of bad players may collude!

- **Elections:** A set of mutually distrusting players, each with a binary preference, wants to compute their majority vote (namely, if there are more 1's or 0's), maintaining the complete privacy of all votes (except what is revealed by the majority outcome).[13]

---

[11]This "impossible" task is what your computer easily performs with, for example, Amazon's computer, over the public Internet, before sending Amazon your credit card number, so it had better work.

[12]Without the last demand, a perfect protocol would instruct them both to hold their tongues.

[13]For example, if three people participate, I voted 0, and the majority outcome is 1, I automatically learn that the other two voted 1.

- **Mental poker:** A set of mutually distrusting parties wants to play a full game of poker (say, by Texas hold'em rules) without cards or other physical means, from the random distribution of hands and betting rounds to the final decision of who won and how much (without revealing anything else about the players' hands or strategies).

If this is not enough, an enormous layer of complications is added if agents are simultaneously engaged in different protocols with different sets of agents, but they are free to use the information from all in their interactions!

Despite this huge variety of tasks and adversaries, an elegant theory was developed that allows both the formal security definitions and the design of protocols satisfying them for essentially any reasonable cryptographic task. Let us slowly describe some of the key ideas leading to this understanding. It should not be surprising that reductions and completeness will play an important role. We begin with the most basic problem of cryptography—secret communication—which we discuss in some detail.

18.5   Probabilistic encryption

There is no better place to start this explanation than the seminal paper of Goldwasser and Micali [GM84] titled "Probabilistic encryption." This paper is the first to introduce precise mathematical definitions of cryptographic security and design protocols satisfying them under hardness assumptions. The language, framework, ideas, and techniques in this paper, (as well as the insistence on the most stringent possible security definitions) became a model for the many thousands (no exaggeration!) of crypto papers that followed.

This paper aims to solve the first cryptographic task in the list above, *secret communication*. The foundational ideas addressing this task, namely the *public-key encryption* of Diffie and Hellman [DH76] and its instantiation based on the hardness of the RSA [RSA78] and factoring [Rab79] functions, were already known. These schemes provide a way for Alice to encrypt her messages so that Bob can decrypt them (and vice versa), while the ability of Eve to decrypt these messages entails an efficient algorithm for the assumed hard problem (which is impossible). But as Goldwasser and Micali point out in their critical analysis, these encryption schemes neither protect against the *complete* decryption of *some* messages by Eve, nor do they protect against Eve obtaining substantial *partial* information about *every* message from their encryption. This paper instead aims to allow Eve absolutely nothing! How to define this? The opening of their paper states it most clearly:

> "This paper proposes an encryption scheme that possesses the following property:
> *Whatever is efficiently computable about the cleartext given the*
> *ciphertext, is also efficiently computable without the ciphertext.*"

This property is formalized in the paper under the name *semantic security* and is suggested as a computational analog of Shannon's information-theoretic notion of *perfect security*, described in his seminal paper on information-theoretic cryptography [Sha49a] mentioned above. Roughly speaking, semantic security demands that for every possible distribution on any finite message space, and for every function f on that space, the following holds: the *a posteriori* probability that Eve computes $f(m)$ after seeing an encryption of a random message m is essentially the same as the *a priori* probability of $f(m)$ over the original distribution.

Of course, in the computational setting, this holds under the Axioms 1 and 2 above. In contrast, Shannon proves that perfect security in the information-theoretic setting *requires* Alice and Bob to have shared information (and so the task as defined is impossible in this setting). Indeed, Shannon proves, using his recently developed information theory, that Alice and Bob must share a random sequence whose length is at least the entropy of the underlying message distribution (this amount is sufficient as well, as the two parties can use the so-called *one-time pad*). In the computational setting, no shared information is needed to achieve perfect security, as the paper [GM84] proves.

This paper also suggests another notion of security, called the *polynomial security*, which is easier to argue about and is proved equivalent to semantic security. Here there is no distribution on the message

space and no auxiliary function to worry about. Simply, for *every* two possible messages, known to Eve(!), she cannot tell their encryptions apart with probability significantly better than random guessing.

It is good to reflect on how these extremely stringent security demands actually force the encryption scheme of Goldwasser and Micali to be drastically different in nature from all past encryption schemes, namely, to be *probabilistic*, as the paper's title proclaims.[14] Consider a message space that contains only two possible messages (say, 0 and 1). As Eve sees all communication between Alice and Bob, she can apply Alice's algorithm to 0 and 1 just as well as Alice could.[15] If encryption is a deterministic function of the plaintext message, as in past encryption systems, then Eve should have no problem distinguishing the two encryptions.

The key idea of Goldwasser and Micali is that these encryptions would be chosen at random, from an exponentially large space! In this case, Eve's strategy is useless. It remains to argue how can Bob figure out the message. The idea behind this first fully secure public-key encryption scheme (omitting many important details) is simple. As in RSA and Rabin's scheme, Bob chooses (as his private key) two random n-bit primes[16] and sends their product M (which is the public key in this system) to Alice. To encrypt 0, Alice sends Bob a random quadratic residue modulo M, and to encrypt a 1, she sends Bob a quadratic nonresidue modulo M. The factorization of M available to Bob makes it easy to determine which is the case, but without this *trap-door* information, Eve is powerless to tell the two distributions apart with higher probability than just random guessing (even though they have disjoint support, and so information-theoretically are easy to distinguish). This *quadratic residuosity assumption (QRA)* was the Goldwasser-Micali instantiation of Axiom 2, but later papers showed how to base such secure public-key encryption schemes on any trap-door function. Note that QRA is a statement about *computational indistinguishability* as discussed at length in Section 7.3, which became a cornerstone of cryptographic security definitions and proofs.[17]

Note that again, "perfect security," despite its name, may turn out not to be always sufficient (or has to be more specifically defined), as some applications may demand stronger notions or extra properties. This is one force that has been driving the field, demanding new definitions and protocols. Examples of such stronger security definitions (and public-key encryption schemes achieving them) that were subsequently developed to answer further constraints include *deniable encryption* [CDNO97], *nonmalleable encryption* [DDN03], and encryption resisting *chosen ciphertext attacks* [NY90], whose names suggest some of these extra properties.

18.6   Basic paradigms for security definitions

After this long discussion of the problem of secret communication and the standards it sets for security definitions, let us consider some of the other tasks mentioned in Section 18.4, and start drawing some general conclusions. These will lead us to the (related) *simulation paradigm* and *ideal functionality paradigm*, whose combination is central for security definitions and proofs. These paradigms evolved in the 1980s and are by now second nature to cryptographers. A recent excellent survey on the subject is [Lin17].

18.6.1   The simulation paradigm

Let us recap, formalize, and then greatly generalize the informal way in which Goldwasser and Micali [GM84] state the security of their encryption scheme: *Whatever is efficiently computable given the ciphertext is also efficiently computable without the ciphertext*. Their protocol (sketched above) specifies the conversation between Alice and Bob if she wants to send him a secret message (say, 0 or 1). Note that the security

---

[14]Note that while randomness is of course an essential part in many past encryption schemes (in particular, all those mentioned above), it was used only for *key generation*. Once a key is chosen, the encryption of any message using that key is *deterministic*.

[15]This uses another basic principle, already codified by Kerchoffs in 1881 (see, e.g., [Kah74, p. 235]), which states that *the compromise of a cryptographic system should cause no inconvenience to the correspondents*. In plain (or modern) words, this means that for security analysis, each participant of a cryptographic protocol can be assumed to know the algorithms used by all others (but not their private inputs or random coin tosses).

[16]n is called the *security parameter* of the scheme, an auxiliary input, according to which computational complexity is measured.

[17]Recall that, not at all coincidentally, the seminal work on computational pseudo-randomness [BM84, Yao82b] was done at the same time and place—University of California, Berkeley, where Goldwasser and Micali were graduate students.

statement does not merely demand that Eve cannot learn Alice's secret message from the conversation, nor that she cannot figure out Bob's private key. It demands that everything she can do after the conversation, she could do without it. And I really mean *everything!* The conversation between Alice and Bob, beyond completely protecting their secrets, can't help Eve solve any possibly unrelated problems, like factoring the integer $M+2$ or finding love. How can one ensure such a strong property? As often happens, proving general statements may be easier than specific ones, and more telling. The following would surely do to ensure it: Let us demand that Eve generates the conversation between Alice and Bob *by herself*, without listening to them. If she can do that, surely listening to them doesn't teach her anything she didn't know!

The meaning of this requirement from Eve is now the key to everything. First, recall that the conversation is probabilistic. Namely, there is a probability distribution on binary sequences (say, D), which is defined by Alice's and Bob's secret inputs and randomness, and the actual conversation between them is one random sample from this distribution. So, a first attempt is demanding that Eve sample from D by herself. This will surely do, but of course is impossible (it is at least as hard as guessing Alice's message bit without any information).[18] The next best thing (which, in our computational world set by Axiom 1, is equivalent), is demanding that Eve sample from any distribution D' that is *computationally indistinguishable* from D. Recall from Section 7.3 on pseudo-randomness that D and D' are computationally indistinguishable, denoted here $D \approx D'$, if no efficient algorithm can tell them apart with probability better than random guessing.[19] Now this is really easy, as Eve doesn't need Alice's or Bob's secrets for that task. She can pick two random n-bit primes by herself, multiply them to create M', the first part of the conversation (so far identical in distribution to that part of the actual conversation). Now what? Simple. She picks a random quadratic residue modulo M' as the second part. Namely, she is simply encoding a 0, as Alice would. This would be fine (indeed, identically distributed) to the actual conversation if Alice's secret message was 0. But what if it were 1? No problem! The intractability assumption (choice of Axiom 2) underlying the [GM84] protocol was precisely that these two distributions, a random quadratic residue and a random quadratic nonresidue, modulo a product of two random primes, are computationally indistinguishable. And thus $D \approx D'$.

Let us now address a subtle issue. How can we "demand" that Eve does anything? Eve is some unknown algorithm that certainly does not cater to our wishes. What this demand formally translates to is the following requirement, which explains the word *simulation*. The security requirement is that for every efficient algorithm Eve, there exists an efficient algorithm (the simulator) we call Eve$'$, that can generate D'. And this simulation constitutes the reduction[20] from the security of the encryption scheme to the intractability assumption. Spelled out, by the intractability assumption, anything Eve could do with D, Eve$'$ can do with D', without any access to the conversation. Which is precisely what the promise "whatever is efficiently computable given the ciphertext, is also efficiently computable without the ciphertext" means.

The task of secret communication is simple from several points of view. First, there is only one (identified) possible adversary, namely, Eve, who is not a real participant in the protocol and has no inputs or information related to it. Second, we have already been given a protocol, and so it was quite obvious what to demand of the simulator; however, cryptographic tasks typically are defined before they are solved. To deal with these issues in more complex situations, we move to the next paradigm.

18.6.2   The ideal functionality paradigm

Let us consider a few of the other tasks from the list in Section 18.4 and see what would be natural to demand the simulator to simulate in each case (focusing on the essence and ignoring many details). Then again we'll generalize it to a paradigm. For each task, we recall its informal definition before discussing the simulation.

---

[18]This follows from the fact that Alice's encryptions of 0 and 1 have disjoint supports.

[19]Again, the most common choices define "efficient" to mean polynomial time in the security parameter n, and "nonnegligible" to mean vanishing faster than inverse polynomial in n. But other choices work as well.

[20]An important issue we will not get into here is how this reduction operates: in most reductions, the simulator Eve$'$ uses the algorithm Eve in a *black-box* fashion. This was believed to be completely general, until Barak [Bar01] found an ingenious non-black-box reduction. Such simulations allowed proving much stronger theorems in various settings.

- **Zero-knowledge proofs:** (see also Section 10.2) Alice attempts to convince Bob that she has a proof of a mathematical statement known to both. If false, then Bob rejects their conversation; and if true, Bob accepts it but learns nothing from it beyond the truth of the given statement.

  Let's deal only with the zero knowledge aspect of this task. (One should also deal with the fact that this is a proof, and in particular its soundness: Alice should not be able to cheat Bob into believing a false statement.) Zero knowledge demands that if the statement is true, Bob does not learn anything from the conversation between them except that truth. What should that mean? As before, we demand that Bob, given a true mathematical statement (say, s), would generate, *by himself*, the conversation he and Alice would have on joint input s (which should be a convincing argument of that truth). More precisely, the security requirement is that for every efficient Bob,[21] there is an efficient simulator Bob' that for every such input s can sample from a distribution that is computationally indistinguishable from the one Alice and Bob would have on s.

  Note that there are two differences between this case and the secret communication example. First, Bob has an input; but of course, Bob' has access to that input as well. Second, Bob is allowed to learn something, namely, that the statement s is true (when it is). This is handled by demanding the existence of a simulator only for such inputs. Another issue regards the following typical confusion. Why do we demand a simulator for *every* algorithm Bob? After all, Bob is now a participant, and his actions are dictated by the protocol (unlike Eve above, who was a "free agent"). The answer is that security demands should take into account all eventualities, specifically, the possibility of players not following protocol. Note that Bob has much more power than Eve above. He actually communicates with Alice. So, if bad, he may decide to ignore the protocol and send messages that may cause Alice to leak information, either about the proof or about anything else Bob didn't originally know and wants to. As before, requiring a simulator for every algorithm Bob ensures that such deviations cannot violate the zero-knowledge property.

  Finally, note that we have only completed the formal security definition for zero-knowledge proofs. How to construct a protocol meeting this seemingly impossible task is a different challenge altogether. A protocol giving zero-knowledge proof for *every* mathematical statement in $\mathcal{NP}$ is described in [GMW91] and is discussed in Section 10.2.

- **Millionaires' problem:** Alice and Bob wish to find out who is richer, without *any* other information about their respective fortunes leaking to the other.

  We are seeking an interactive protocol for which Alice and Bob, on any (respective) inputs x and y (say, integers between 1 and M), to learn $GT(x, y)$ (a bit defined to be 1 if x is greater than y and 0 otherwise) but nothing more. The security definition is again similar here. Let us spell it out. Here each of the players should have a simulator. For every efficient Alice, we'd like an efficient simulator Alice' that does the following. On any input x and a bit b, Alice' samples from a distribution that is computationally indistinguishable from the conversation Alice would have with Bob on any y satisfying $GT(x, y) = b$. The demand for Bob's simulator is analogous. Again, a protocol will specify algorithms for Alice and Bob. But by demanding a simulator for every possible efficient algorithm that may be "impersonating" one of them, we achieve security even if either a "fake" Bob or Alice deviates from the protocol. We note that an interesting issue in this task (and many others), which we ignored here, is how to define security (and even the output value) if one of the parties terminates communication before completion, namely, before $GT(x, y)$ was computed by both.

  As before, this is just a security definition, and finding a protocol meeting this impossible-looking task is a nontrivial challenge, even for perfectly honest players! Think about it. A protocol for this task was given by Yao in [Yao86], and we will discuss it shortly.

- **Elections:** A set of mutually distrusting players, each with a binary preference, want to compute their majority vote (namely, if there are more 1's or 0's), maintaining the complete privacy of all votes (except what is revealed by the majority outcome itself).

---

[21]Namely, someone interacting with Alice as Bob, who may not follow the protocol.

This is getting complicated, so we'll ignore many issues, like preventing participants from voting twice. We focus on privacy: What should be the formal meaning of the intuitive statement that privacy of the votes is preserved after the majority value is revealed to all? For example, suppose there are five participants, two vote 0, three vote 1, and so the majority value is 1. Let us consider a few scenarios to arrive at a plausible definition of privacy. Suppose the two 0 voters get together at the end of the protocol and reveal to each other their vote. They immediately conclude with the perfect knowledge that each of the other voters must have voted 1. Does this violate the privacy requirement? It clearly doesn't, as this would happen even in an ideal world, in which a trusted party (with a secure private channel to each player) would collect all votes, compute the majority, and announce it to all. In contrast, if two of the 1 voters got together and revealed their votes to each other, all they can conclude is that there is at least one more 1 vote among the remaining 3 participants (which they would learn even in this ideal world), but must not know anything more. There are plenty more scenarios even with five players, let alone more. Should we list all these possible scenarios in the privacy requirements of an election protocol? The answer is no!

There is a very concise way to describe all these conditions, which is suggested by the hypothetical trusted party mentioned above. The *ideal functionality paradigm* asserts that the privacy in the real protocol (where there is no trusted party) should strive to match that of the ideal protocol (in which the trusted party communicates privately with each player and solves the problem for them). For elections, this generalizes the Goldwasser-Micali maxim for secret communication to the following security requirement: *What a subset of players learns by participating in the election protocol, it could learn from participating in an election with a trusted party.* As in the original, this statement must hold for all possible inputs to the players. So now the analogous security definition, for any fixed subset of players, is the following: For any set of efficient algorithms these players run, there exists an efficient simulator, which for every input value to this set and for a given majority outcome (hypothetically received from the trusted party), samples from (a distribution indistinguishable from) the actual transcript of the protocol. A protocol achieving such security, for all subsets comprising less than half of all players,[22] was developed in [GMW87] and will be discussed soon as well.

I hope you see a pattern forming here—this paradigm captures neatly and perfectly the other (simpler) examples we have seen so far. The security requirement for zero-knowledge proofs is what you get if a trusted party received an alleged proof from Alice for the statement s, checked it, and told Bob whether it does or does not constitute a valid proof. Similarly, the security requirement for the millionaires' problem is what you get if Alice and Bob respectively send x and y to the trusted party, who computes $GT(x, y)$ and announces the answer to both.

- **Mental poker**: A set of mutually distrusting parties wants to play a full game of poker (say, by Texas hold'em rules) without cards or other physical means, from the random distribution of hands and betting rounds to the final decision of who won and how much (without revealing anything else about the players' hands or strategies).

  We ignore many complications and details here, and focus on the high-level message. This cryptographic task is far more complex than voting. Nevertheless, the ideal functionality paradigm should guide us to the formulation of a security definition. All we have to imagine is a trusted party with a secure channel to each player who is running the show according to the rules of the game. Given this specification, the security requirement for the actual protocol (for any particular subset of players) is that anything they learn by participating in the real protocol, they could already have learned in that ideal implementation. And again, demanding a simulator for that subset would guarantee that requirement. A two-party poker protocol follows from [GM82, Yao86], and for any number of players in [GMW87].

Summarizing all we have discussed, the ideal functionality paradigm together with the simulation paradigm make an extremely convenient framework for formalizing the most general security definitions of complex

---

[22]In a certain sense, this is the best one can hope for.

cryptographic tasks. Now that we have dealt with this central issue, let us turn to the challenge of designing protocols meeting these security definitions.

18.7   Secure multi-party computation

After protocols for the many diverse cryptographic tasks were designed in succession in the late 1970s and early 1980s, Yao suggested that a single task may capture most of them. He formulated it for tasks with two players in [Yao82a] and described an ingenious protocol for it in [Yao86]. This result was soon extended to any number of players by Goldreich, Micali, and Wigderson in [GMW87]. The task Yao proposed came to be known as "secure multi-party computation" (SMC),[23] defined as follows.

- **Secure multi-party computation:** Some players wish to compute a public function f on their private inputs. An explicit description of the function as a Boolean circuit[24] is known to all of them, but each of the input bits to the circuit is known to exactly one of them (a player can own many bits). The value of f on these inputs, but nothing else, should become available to all.

It is completely clear that most examples in this chapter so far fit this description (indeed, the millionaires' problem and the election were explicitly defined as function evaluations: GT and Majority, respectively). Others require a simple modification. For example, collective coin flipping and other probabilistic functions can be modeled by allowing the circuit to have random inputs (which the players will have to generate together). Yet other tasks, like the poker example and other "mental games," require a more substantial modification of SMC due to their interactive nature. One can also accommodate the natural extension that different players compute different functions of the same inputs (as is the case in the secret exchange problem). All of these extensions fall under SMC, but for simplicity, we will only refer to the deterministic, static, single-output definition given above.

The extremely general SMC problem, capturing almost any imaginable cryptographic task, has a *secure* protocol, where security is defined by using the general paradigms above. Namely, what the players experience is identical to sending all their inputs to a trusted party, who evaluates the circuit and sends them back the correct answer. The single behavior possible in the real protocol but not in the ideal one is early abort (or failure) by some players, which may cause others never to see the output. This cannot be helped in 2-player protocols when one player aborts (or more generally, if half the players abort). But we will see that if a majority of the players follows the protocol, we can achieve *fairness*, namely, the guaranteed output delivery to all players who did not abort.

The intractability assumption underlying this general protocol is the most *standard* hardness assumption: the existence of trap-door functions (the same one used for the secure communication protocol, which is quite close to the latter in power).[25] Thus, essentially, *if the most basic cryptographic task (namely, secret communication) has a secure protocol, so does the very general SMC task* (we shall return to this important point).

It gets even better: *protocol design becomes completely automated*. There is an efficient algorithm that, when given as input the circuit to be evaluated by the parties, outputs a description of the secure protocol (namely, a specification of the algorithms of the participants). I state here an informal version of this completeness theorem. Let us distinguish (as is common) *fair* secure protocols (in which the delivery of the output to all participants is guaranteed) from merely just secure protocols (in which early termination of some players can cause other players never to learn the output value.)

Theorem 18.1 [Yao86, GMW87]. *Assuming trap-door functions exist:*

---

[23]Sometimes also called and abbreviated differently in the literature, for example, "secure function evaluation" (SFE) and "multi-party computation" (MPC).

[24]Recall that Boolean circuits are a universal model of computation, as discussed in Section 5.2. Thus, any efficiently computable function can be encoded by a small circuit.

[25]The relationship between the trap-door function primitive and the secure communication primitive is actually quite complex. The advanced reader may want to consult [GKM$^+$00, HO13] on the interrelationship of these and many other cryptographic primitives.

- *SMC has a secure protocol for any number of players $n \geq 2$.*

- *SMC has a* fair *secure protocol for any number of players $n \geq 3$, provided a strict majority of the players follows the protocol.*

I now informally and at a very high level highlight some of the key ideas that go into the design of a protocol for this general problem. Note that the full proofs are extremely involved and subtle. The original papers are missing many details; full details can be found in Chapter 7 of the second volume of Goldreich's book [Gol04].

The proofs are divided into two conceptually separate parts. To explain them, let us discuss the way in which bad players can misbehave. In the security definitions above, we put no restrictions on the adversaries (besides efficiency, as per Axiom 1); a simulator was required for every possible adversary (representing a participant or a subset of participants). Such adversarial behavior is called *malicious* in the literature, and it is these most general adversaries we want security against. A far more benign type of adversarial behavior is called *semi-honest*. Like *honest* players (the good guys), who follow the protocol specification verbatim, semi-honest do so as well, and the only fault each is allowed is simply to break down, namely, stop all communication at some arbitrary point during the execution of the protocol (which may depend on the execution). Intuitively, such adversaries should be much easier to handle, but note that some of the problems above look impossible even when players are completely honest (e.g., the millionaires' problem, zero-knowledge proofs). The two parts of the proof show:

1. how to design an SMC protocol secure against (any number of) semi-honest adversaries, and

2. that *every* protocol secure against semi-honest adversaries can be enhanced to become secure against malicious adversaries and can be made fair as long as the latter are the minority.

The beauty of both parts is that each uses the main tenets of computational complexity, namely, completeness, reduction, and locality of computation. Both parts use locality to identify a simple, *complete* component (which itself is a very special case of the same problem), design a protocol for this special case, and then use reduction (or composition) to extend it to the general case. Note that all this is possible partly *because* the SMC problem is general enough—it may be much harder to do reductions between protocols that only work for special cases of SMC (e.g., the millionaires' problem). I now state more precisely the results of each of these two parts and sketch the ideas for proving each.

A note on encryptions. Both parts rely on (different) encryptions of the players' individual inputs, which have special properties. I will not specify their nature or how to achieve them, and only note that both types can be easily constructed from standard trap-door functions. I will simply assume that they exist and explain how they are used.

Start with part (2), which was already obtained a year before the SMC results in [GMW91]. Recall that this is the paper mentioned in the zero-knowledge Section 10.2. One major result of this paper is that every $\mathcal{NP}$-statement has a zero-knowledge proof. In the same paper, they applied this idea to simplify protocol design.

Theorem 18.2 [GMW91]. *Every protocol secure against semi-honest adversaries can be enhanced to become secure against malicious adversaries, and it can be made fair as long as the latter are a minority.*

We reduce the theorem to a sequence of zero-knowledge proofs. Let me elaborate. How can we force a potentially malicious party, say, Alice, to follow the protocol? Simply by verifying that each message Alice sends is computed according to her (publicly known) specified algorithm. Well, that algorithm instructs that the message at hand should be a (publicly known) efficiently computable function, say, g, applied to some publicly known data x (e.g., past messages in the execution of the protocol) and private data y known only to Alice (like her inputs and private randomness). Thus, Alice sends some message m, and the other players wish to verify that $m = g(x, y)$, without Alice revealing y. Assume that every player publishes, at the beginning of the protocol, a *commitment* of its private data (e.g., an encryption of it using its own public key). Thus, all players have access to $z = c(y)$, Alice's commitment to her private data y (the commitment

function c is public as well, of course). But now the claim $m = g(x, y)$ becomes an $\mathcal{NP}$-statement! The $\mathcal{NP}$-witness is simply y itself (which includes Alice's private randomness); having access to y (and the publicly available x), one can efficiently verify both that $z = c(y)$ and that $m = g(x, y)$. This means by [GMW91] that there is also a zero-knowledge proof of these equations, which Alice can now conduct as a prover, with the other players as verifiers. This of course needs much elaboration, which I will not give here (e.g., how other players are coordinated, how to avoid infinite recursion if some of them are malicious).

Note that in part (2) above, "locality of computation" was simply the fact that a protocol execution is just a sequence of messages. We now move to part (1), where locality of computation will refer to the fact that every circuit evaluation is a sequence in the evaluations of Boolean gates.

Theorem 18.3 [Yao86, GMW87]. *There exists an SMC protocol secure against (any number of) semi-honest adversaries, which is fair as long as a majority of them do not fail before the end of the protocol.*

Yao's proof for 2-party SMC reduces the design of a protocol for arbitrary circuits to the design of a protocol for a single gate.[26] The key insight was a definition of gate evaluation on hidden inputs in a way that composes with evaluating arbitrary circuits. And the idea is natural: Simply ask for the output to be hidden as well. The main problem is that different inputs are owned by different participants and are hidden from others. Thus, one needs to define "hidden" in a way that is consistent for all inputs and outputs of all gates. Then, if one gate can be so evaluated "in the dark" by the players, they can iterate the process and evaluate any circuit. Yao describes such an ingenious encryption scheme (which I will not explain), in which a gate can be evaluated by a single player. The second player just has to encrypt its inputs in this format, which is done using Rabin's *oblivious transfer* protocol [Rab81, EGL85]. For the multi-party case, [GMW87] uses a distributed encryption, in which the hidden values are *shared* among all players, who also carry on the gate-by-gate evaluation *jointly*. The joint construction of these distributed encryptions is performed via Shamir's secret sharing scheme [Sha79b]. Indeed, we need the *verifiable* variant of secret sharing due to [CGMA85] to protect against early abort. The gate-by-gate evaluations from these encrypted shares can now be performed by any majority of the players. Finally, in both protocols, there is also a way for (any majority of) the players to decrypt the final output of the circuit and make it available to all.[27]

Let us conclude the SMC story by noting that the general theorems discussed are far from the last word. One important extension [CDD+99] was proving them secure for an *adaptive adversary* (one who observes the execution of the protocol and can corrupt different players according to it). Another important extension [Can01], through the paradigm of universal composability (which applies to protocol design beyond SMC), is proving them when the protocol is not executed in isolation but rather concurrently with others, related or unrelated, involving the same players and possibly others.

18.8   Information theory vs. complexity theory: Take 2

Most of this chapter has hammered home the point that many tasks that are provably impossible in the information-theoretic setting (when no limitations are put on the computational power of players) become possible in the complexity-theoretic setting (when they are so limited). Now I stress the surprising lesson that one should rethink the seeming weakness of the information-theoretic setting in light of the power of complexity-theoretic protocols. The shining example is the secure multi-party computation (SMC). Recall the informal conclusion we drew from the fact that SMC can be based on trap-door functions: The ability to perform the basic task of secret communication entails the ability to perform the vastly general SMC. This idea can now be considered abstractly. Assume we have perfectly secure communication channels between every pair of participants. This assumption guarantees *information theoretically*, the basic task of secret communication by definition. What other tasks can be performed in this model with information-theoretic security? The answer turns out to be, as in the computational setting, practically all of them! Shortly after the computational papers on SMC, protocols for SMC were designed in this information-theoretic setting.

---

[26]It may be amusing to realize that this problem, say, for the AND gate, is the very special case of the millionaires' problem in which each of their fortunes is restricted to 1 bit. This problem is still highly challenging, even if no one cheats!

[27]Again, in the case that half or more of the players abort, some others may not get the output (which is unavoidable, as e.g., for two players).

This was done in two independent simultaneous works, using different techniques, one by Chaum, Crépeau, and Damgård [CCD88] and the other by Ben-Or, Goldwasser, and Wigderson [BOGW88] (see full details in [AL11]).

Theorem 18.4 [BOGW88, CCD88]. *Assume private communication channels between every pair of players. There is a protocol for SMC that is secure against computationally unlimited adversaries as long as more than 2/3 of the players are honest.*

Note that here we can only tolerate $< 1/3$ cheating players, whereas the computational setting could tolerate $< 1/2$. However, as in that setting, there is a natural sense in which the constant $1/3$ is the best possible in this one. Of course, the security guarantee is far better, and the assumption may be reasonable in many concrete settings. Also, the protocols are simpler and significantly more efficient than in the computational setting. Finally, note that these protocols provide a completely different proof of the (fair) computational SMC results (and with the weaker constant), essentially by replacing the assumed secure private channels by a computational encryption scheme.[28] In other words, had the information-theoretic protocol been discovered earlier (of course, it could have been), it would have yielded the computational result using only cryptographic secret communication. These connections between the two settings, information-theoretic and complexity-theoretic, are both important and fascinating. These connections are manifested in other ways and places,[29] and will surely play an important role in future developments.

18.9   More recent advances

Until now we have talked about very old results that shaped the field. However, cryptography is an extremely dynamic field, constantly reshaped by changing demands from the real world, as well as by the emergence of new ideas and techniques that resolve very old problems and challenges. In this section, we discuss such exciting recent developments.

18.9.1   Homomorphic encryption

Homomorphic encryption is a public-key encryption scheme that allows anyone in possession of the public key to perform operations on encrypted data without access to the decryption key. Initial public-key systems that allow this for either addition or multiplication (over appropriate rings), but not both, were known as soon as such systems emerged, in the late 1970s. The challenge to create a system in which both operations are simultaneously possible, called a *fully homomorphic encryption (FHE)* scheme, had already been raised then [RAD78]. It was also clear at the time what a powerful cryptographic primitive FHE would be, and more applications for it emerged over time.

As one application, you could delegate computation on sensitive data to others. For example (this may be irrelevant for some readers), you can homomorphically encrypt your financial information, let your accountant do your taxes "in the dark," and then decrypt. The importance of such an ability became enormous with the advent of (computationally weak) mobile devices and (computationally powerful) cloud services, which routinely perform computations for them. FHE guarantees complete privacy of the inputs to these computations (and much more, as we shall see when discussing delegation). As another application, SMC, which captures numerous cryptographic tasks, could be achieved rather simply using FHE, and with a huge bonus. Namely, the known protocols, which are very communication intensive, can be replaced (aside from pre- and post-processing) by a minimal communication protocol: The only communication needed is sending the encrypted inputs and outputs before and after the computation, respectively.

Indeed, FHE looked just too good to be possible, but like other impossible-looking cryptographic tasks, it has yielded as well. It took 40 years for the construction developed in the seminal PhD thesis of Gentry [Gen09a, Gen09b]. Gentry's ingenious initial construction was very complex and costly, and its security

---

[28]Of course, they give nothing for $n = 2$ players.

[29]For example, recall one beautiful example we have seen in Section 9.2: the construction of *information-theoretic* extractors from *complexity-theoretic* pseudo-random number generators by Trevisan [Tre99].

was based on the hardness of a somewhat nonstandard version of lattice problems. As expected, vigorous work ensued to simplify the construction, make it more efficient, and weaken the hardness assumption to a more standard one (as well as implement FHE in practice). One such advance is the paper [BV14], which provides considerable simplification and relies on the more standard learning with errors assumption of [Reg09, Pei09]. It is a challenge to base FHE on general assumptions, like the existence of trap-door functions.

18.9.2   Delegation of computation

Let us elaborate on the "cloud computing" application, which has numerous incarnations today. A weak computational device Alice (e.g., a mobile phone or a laptop) wishes to perform a computationally heavy task, beyond her computational means. She can delegate it to a much stronger (but still feasible) machine Bob (the cloud, or a supercomputer) who offers the service of doing so. The problem is that Alice does not trust Bob, who may give the wrong answer due to laziness, fault, or malice. Can Bob convince Alice that the answer is correct much faster than it takes her to compute the answer herself?

This is clearly a version of the interactive proof setting discussed in Section 10.1, but with two essential differences in the efficiency parameters, which are now "scaled down" to accommodate the application to delegation. The following *doubly efficient* interactive proof requirements were formulated by Goldwasser, Kalai, and Rothblum [GKR08]: We want *efficient* provers and *highly efficient* verifiers. More specifically, assume that Alice wants to compute $f(x)$ for an input x of length n, where f is described by a known circuit of much larger (but still polynomial) size. The prover, Bob, should run in polynomial time[30] (as opposed to having unlimited time in standard interactive proofs). The verifier, Alice, should only run in (nearly) linear time (as opposed to polynomial time in standard interactive proofs). Another feature of this setting that differs from the standard interactive proof model is that some protocols have very few rounds of interaction, often as low as only one round: one message from Alice to Bob, and one from Bob to Alice.

After initial progress [GKR08, TKRR13], the problem was completely solved in the beautiful paper of Kalai, Raz, and Rothblum [KRR14]. We state their result below, and note that more recent works studying the power of the prover in interactive proofs for other classes below $\mathcal{PSPACE}$ develop the framework of *doubly efficient proofs* (see e.g. [RRR16, GR18]). In a different direction, this idea was used for verifying quantum computation (under appropriate cryptographic assumptions) in [Mah18].

Theorem 18.5 [KRR14]. *Assuming fully homomorphic encryption, there is a 1-round doubly efficient interactive argument for every computation.*

One of the remarkable ingredients in this (cryptographic) Theorem 18.5 is a seemingly unrelated result in the standard (noncryptographic) model of interactive proofs. Recall from Section 10.1 that in multiple-prover interactive proofs ($\mathcal{MIP}$), a verifier is allowed to interact with several provers, who are not allowed to communicate. This paper introduces a variant of this model, called *no-signaling* $\mathcal{MIP}$,[31] and completely determines the languages it proves: the class $\mathcal{EXP}$ of problems solvable in deterministic exponential time. Compare this with Theorem 10.4, which for the original version proved $\mathcal{MIP} = \mathcal{NEXP}$.

Going back to the original delegation question, note that there is another sense in which Alice may not trust Bob—she may want to protect the privacy of her input x from him, but still wants him to perform the computation for her. This is the model of private information retrieval (PIR), introduced in [CKGS98]. The reader is encouraged to discover the exciting developments and remaining problems regarding PIR, which we will not discuss here. Also note that the delegation Theorem 18.5 need only assume PIR, as opposed to the (seemingly stronger) homomorphic encryption.

18.9.3   Program obfuscation

Now here is a really strong cryptographic primitive, which every software company would love to have: a *program obfuscator*. An obfuscator O (which is allowed to be probabilistic) is an efficient algorithm that

---

[30]Such interactive proofs are sometimes called *arguments* in the literature.

[31]This no-signaling was first studied in the context of quantum interactive proofs—it is a notion that comes from physics, motivated by the fact that information cannot travel faster than light.

maps Boolean circuits to Boolean circuits in a way that completely hides *every* aspect about them except their functionality. More precisely, if it maps a circuit C to a circuit $O(C)$, then $O(C)$ computes precisely the same function as C does, but reveals no more information about C than black-box input-output access to C would. For example, a company that designed a game or other piece of complex software based on their secret technology and know-how can release the obfuscated code without fear of their competitors. Program obfuscation is a fantastic kind of encryption that allows you to use a program code but not to understand how it works. Its existence would imply numerous other cryptographic primitives, including most of those discussed in this chapter.

Unfortunately, program obfuscators cannot exist. This was discovered (with a highly complex proof) in the same paper by Barak et al. [BGI$^+$01] that formally defined this notion. Given this sad news, they suggested an alternative definition, called an *indistinguishability obfuscator* (or IO). An IO is also an efficient probabilistic map from circuits to circuits that preserves functionality (namely, $IO(C)$ computes the same function as C), which satisfies the following (indistinguishability) property. For every two *functionally equivalent* circuits of the same size C, C', their obfsuscations $IO(C), IO(C')$ are computationally indistinguishable.

Two things are unclear at this point. Can one construct IO (under reasonable intractability assumptions)? And what is it good for? These questions remained unanswered for more than a decade, and then, starting with the breakthrough paper of Garg et al. [GGH$^+$13], the subject came to life again. Indeed, the activity regarding this topic brings all the excitement of an excellent thriller, with ingenious constructions of IO based on new, unfamiliar hardness assumptions (mainly group theoretic and number theoretic), new algorithmic attacks refuting some of these assumptions (e.g., see [CGH$^+$15, HJ16]), improved and repaired assumptions immune to past attacks, raging arguments about the reasonableness of surviving assumptions, and more. And this is only regarding the first question, of the existence of IO. On the second question, of the utility of IO, there has been a flurry of activity as well, revealing its potential power. It was shown to imply a host of other extremely useful cryptographic primitives, some of which have no previous constructions (e.g., see [SW14, GGHR14]). Furthermore, IO was shown to be equivalent to the "best possible obfuscation" definition of [GR07], and more.

It will be extremely interesting to see how this subject turns out when the dust settles!

18.10   Physical attacks

This section should damp down the uplifting feeling of the wonderful mathematical theory discussed in the previous section. As you know, cryptographic systems abound. Are they secure? Well, we know they are mathematically secure (e.g., since no one has found an efficient factoring algorithm yet, to the best of my knowledge).[32] So, how come we keep reading about hackers breaking into the most secure computer systems imaginable, like the Pentagon's, and about electronic theft from bank and credit card accounts? Of course, the answer is that these attacks work outside of the mathematical models above. The *implementation* of security systems offers numerous inroads and loopholes that allow an attacker to invade a system without violating any of the mathematical, rigorous security proofs it rests on! Often, these weaknesses arise for intellectually boring reasons, like stupid and negligent actions of sending passwords or other identifying information in the clear, leaving the door open and the computer on, and so forth.[33] However, research in the past decade or two reveals that even in seemingly watertight situations, minute weaknesses exist that allow ingenious attacks to completely break cryptographic systems.

What might such minute weaknesses be? Computer systems are physical objects, and when they operate, they emit a host of physical signals of various types. Furthermore, these signals are correlated with the operations they perform. In the words of the great cryptanalyst Adi Shamir, *computers are telling us what they are doing, and we just have to listen.* Taking this motto literally, Shamir and collaborators [GST14] have recently devised an *acoustic attack* in which a cheap microphone in the vicinity of a laptop performing

---

[32]And the efficient *quantum* factoring algorithm is still waiting for a quantum computer that can execute it—see more on quantum computing in Chapter 11.

[33]One popular jargon word for obtaining sensitive information this way is *phishing*—the reader can read about such attacks and protecting against them by searching on this word.

RSA decryption can fully extract its secret RSA keys! This is but one example of a quickly developing field of *side-channel attacks* (or simply, *physical attacks*) within cryptanalysis that sends waves of wake-up calls of different forms to designers of security systems to rethink their designs, in an endless game of cat and mouse. Such attacks include (I did not make these up—all phrases taken from actual papers) timing analysis, electromagnetic attacks, optical attacks, power analysis, bug analysis, microwave attacks, cache attacks, thermal analysis, cold boot attacks, and more. Some of these attacks apply to hardware as well! Many require, beyond ingenuity and insight, also sophisticated mathematical and physical analysis. Curiously, many attacks on cryptosystems are discovered in a way that may be likened to the way cures to diseases are discovered: One observes a system under a variety of conditions, until some strong signals appear.[34]

Practically, for the time being, it seems that most such physical attacks cannot be performed on a large scale, namely, applied simultaneously on a million computers, say. Mathematically, the wealth of such physical attacks (many of which are only discovered experimentally) raises the challenge of theoretical abstraction and modeling that may inform the design of better security systems.

18.11   The complexity of factoring

To add another bucket of cold water to the sobering news of the previous section, we go back to mathematical attacks. Many computer security systems today,[35] supporting a vast electronic commerce and financial transaction network, rely on the assumed difficulty of factoring large integers. I believe that most people, and most decision makers, are not aware that a *single*, simple to state mathematical problem, which could potentially be solved by a math or computer science undergrad tomorrow, is the *only* barrier to a potential economic meltdown. Indeed, it is hard to think of any other concrete potential *mathematical* discovery that can cause catastrophic damage of similar magnitude. Are we ready for this eventuality? Sure, we have (very) few alternative trap-door functions that can in principle replace factoring as the basis of these security systems, but these seem much less efficient, and would probably take months or years to implement. What would happen in the meantime? And if you find this doomsday contemplation too depressing, here is another question to mull over. Suppose that you find an efficient factoring algorithm—an amazing discovery that could bring you fame and fortune if published, but could also wreak havoc on electronic commerce. What would you do with it?

---

[34]For example, the paper [GST14] describes how viewing sonograms of a computer when performing a variety of different arithmetic operations revealed patterns of frequencies that correlate with the binary representation of the secret RSA keys as these are processed during the execution of a decryption algorithm.

[35]Perhaps the majority still, even with the rise of elliptic curve cryptography (see e.g. [HMV06]), for which the questions we raise here can be studied as well.

19 Distributed computing: Coping with asynchrony

Distributed computing captures any situation in which many computers or processes aim to achieve certain goals (common or separate) in a way that necessitates interaction. This of course includes server farms; supercomputers; sensor networks; organizational intranets; and the Internet; but also your single laptop, in which many components have some independence. Moreover, it can serve as a model for numerous phenomena in the natural sciences. Clearly, this is a vast and heterogeneous field.

Distributed computing is similar to cryptography in many ways, including the interactions of many parties, the heterogeneity and multitudes of objectives, and the complexity of formal modeling, but the issues central to each are quite distinct. Indeed, there is plenty of interaction; cryptography is used in some distributed algorithms, and distributed algorithmic ideas are exploited in cryptographic protocols.

In this chapter, we focus on one major issue that is present in distributed computing and absent from all other topics discussed in this book, namely, *asynchrony*. Parties[1] in asynchronous distributed computation cannot assume a *common clock* for keeping time and coordinating the global progress of computation among them. The information one expects from another party may be indefinitely delayed, so there is no way to find out whether such a party is simply slow, completely broken, or maliciously refusing to cooperate at this point. Even when all messages from other parties eventually arrive, their order of arrival may be arbitrary. But life must go on, and each party must keep making decisions according to its local view and continue the computation. Note that asynchrony is a real headache: It can arise from computational differences among the parties, properties of the underlying communication between them, or other reasons. It must be dealt with. Indeed, this issue is so central that even the subfield of asynchronous distributed systems is very large. Some excellent texts on this subject include [Lyn96, AW04], which in particular make formal the high-level language discussed below.

We will discuss two of the most basic problems that asynchronous systems need to solve, one in which the parties have a common goal, and one in which they have conflicting goals: *coordination* and *sharing resources* or, in the picturesque language of the field, the *Byzantine generals problem* and the *dining philosophers problem*. We will see that in very general settings, these two problems[2] are actually completely *impossible*. Here, unlike most of the book, where we focus on easy vs. hard, "impossible" really means impossible, as in Turing's result for the halting problem. Namely, there are no *finite* algorithms for these problems! As with Turing's work, impossibility results are extremely useful—important problems really need to be solved, and impossibility results focus on the need to add assumptions or weaken the requirements, as befits a variety of practical settings. In distributed computing, impossibility proofs are very delicate, and the precise definition of what "asynchrony" means and what the communication medium allows are extremely important—we will explore some of the principles underlying these modeling issues. We will also see how an a priori quite unexpected mathematical field—*algebraic topology*—seems to be intimately connected to such distributed problems in asynchronous environments and is very powerful in proving both impossibility and possibility results. Finally, the impossibility results we discuss will be for *deterministic* programs. As we saw in other chapters, randomness is extremely powerful here as well: In the asynchronous setting, it can make the impossible possible!

19.1 High-level modeling issues

Here we informally discuss some major issues that must be formally defined for asynchronous distributed computing. The problems and results we discuss later in the chapter will make some of these issues more concrete. This section is mainly to impress the reader with the complexity of modeling in this field. Again, the textbooks cited above and the references cited give far more precise and detailed information.

---

[1] It must be a coincidence that parties, processors, players, participants, philosophers, P_i, and so forth, all start with the letter "p." We shall use these terms interchangeably in this chapter when referring to the interacting entities in distributed systems.

[2] Hundreds of other impossibility results, as the title of these papers proclaim, can be found in [Lyn89, FR03].

Atomicity   In the absence of a global time that all players can reference, *simultaneous* actions can destroy information (or worse, your assets and even your life in some applications of concurrent algorithms). This is resolved by introducing *atomic* actions. I know, it sounds even worse for humanity, but you know what I mean by "atomic" here: happening instantaneously, without interference. More precisely, certain access primitives of processors to objects (reading from or writing into a memory register, testing for the availability of a resource, seizing control of an available resource, etc.), which in reality are never instantaneous, are treated as if they were instantaneous: Access of an object by one processor must be completed before another can access it. The choice of access primitives naturally varies, depending on the application, and their physical realizability and cost (in hardware or software). The study of their relative power is an important part of this theory (and occupies voluminous literature). I will specify the choices made for the problems we discuss, but just to impress you with the variety, here are the names of some of the atomic *read-modify-write* primitives invented and studied for accessing a memory register: *test-and-set, fetch-and-add, compare-and-swap*, and *load-and-store*.

Program   Concurrent programs (or distributed protocols) assign a program to each participant, whose steps alternate between local computation and some atomic action to shared variables. There is typically no limitation on a processor's computational power or memory or communication.[3] In some but not all problems, an input is given to the participant before the computation starts.

Symmetry   A collection of important problems studied are *symmetric* problems, in which the specification of correct behavior or legal output is the same for all players (who are considered identical, e.g., have no unique IDs). For such problems, it is desirable to have the individual programs used by each player be *identical* (such concurrent programs are sometimes called *anonymous*). We will require it here as well, as the problems we consider are symmetric in this sense.

We now discuss the *execution* (or *run*) of concurrent programs in an asynchronous environment.

Asynchrony   Given atomicity of action, let us articulate the meaning of "asynchrony". We (and the general theory) usually take the worst-case scenario: A (scheduling) adversary controls the execution of a concurrent program. It "wakes up" the different parties at will. It is allowed to schedule and interleave atomic actions in an arbitrary way, with full knowledge of the global system state so far. The only limitation on this adversary is that it must keep each processor's own actions according to the order they were performed; this guarantees that every local view is consistent. However, it is entirely possible, for example, that two successive messages sent from A to B will arrive in the reverse order.

Faults   An important aspect of the theory are the *faults* that a computation must be able to tolerate. As in cryptography, the theory generally distinguishes two major kinds of faults. The benign kind is simple termination (sometimes called "fail-stop" or "crash" fault) of a processor at some point during computation. This can be modeled by having the scheduling adversary indefinitely delay all the future actions of that processor. The malicious kind (typically called "Byzantine") allows a processor to act arbitrarily, completely ignoring the program it should execute. We will focus here on the first kind, again making the impossibility results stronger.

Communication medium   There are quite a few communication models studied in distributed systems. The most common are *message passing* and *shared memory*, each comprising many variants and submodels. In the first model, the players are vertices in an undirected graph, and they can only communicate by exchanging messages with their direct neighbors (see, e.g., [AW04] for formal definitions and background). In the second, there is a set of shared registers supporting some kind of atomic actions. Shared resources

---

[3]This is especially convenient for our focus on impossibility proofs, as it makes them stronger. In actual algorithms, one of course cares about minimizing all of these, see, for example, [EGSZ16].

often provide such means of communication—they are either "locked" (when in use), or "free" (when not), and this status can be accessed and provides external information to processors. The variant we will use when discussing consensus allows each shared register to be read by all, even simultaneously (as in a broadcast setting), but each can be written to by only one specified player.

The shared-memory model often provides a more global view to the participants than the message-passing model.[4] For example, there is no way in the shared-memory model for one party to send different messages to others; everything she writes can be seen by all others. Indeed, it is easy to simulate message-passing algorithms in the shared-memory model (using read-write registers). It is thus easier to design concurrent algorithms and harder to prove impossibility results for the shared-memory model. However the paper [ABND$^+$87][5] gives a simulation in the other direction (namely, of read-write registers in the message-passing system), provided that a majority of the participants do not crash (or alternatively, if less than a third are Byzantine).

Properties of programs   Given the complexities that concurrent programs have to endure (especially the power of the scheduling adversary), it is nontrivial to specify what a "successful" program is. Of course, this depends on the problem it is supposed to solve or the behavior it should induce. I mention, in rough terms, a few of the many properties studied; we will see some in action soon. More precise information on these properties and their relations can be found in [HS11]. Essentially, all require the continued functioning (in various senses) of a system despite arbitrary delays and possible failures of some of its components. In a *wait-free* program, every processor that remains alive (does not fail) must compute its output after a finite number of steps. In a *deadlock-free* program, unless all processors halted, *some* processor will gain access to a requested resource. In a *starvation-free* program, *every* processor that requests some access eventually gets it. In a *fair* program, *every* access granted to one processor will be eventually granted to another who requests it. Note that some properties are ordered: for example, fairness implies in particular no starvation, which in turn implies no deadlock.

It is worth stressing that *a property is satisfied by a concurrent program if it is satisfied in every possible execution of that algorithm (for all possible scheduling of actions and allowed failures).*

19.2   Sharing resources and the dining philosophers problem

Edsger Dijkstra, who laid the foundations of distributed computing and concurrent programming[6] (as well as other areas of computer science) liked to invent stories that capture the many issues arising in the field. The *dining philosophers problem* is among the most famous of these stories and has served as a model problem for concurrency control problems ever since. Dijkstra described it in [Dij71], and Hoare formalized it in [Hoa78]. This problem abstracts the difficulties of resource allocation in symmetric environments. We repeat Dijkstra's informal version, which of course requires translation to the dry language of processors and resources. It is depicted in Figure 20.

Imagine n philosophers sitting around a table. The philosophers are *identical*, namely, have no unique names or IDs. Between every two philosophers is a fork. In front of every philosopher is a bowl of (an unlimited amount of) pasta. Bowls are private, but forks are shared. Philosophers alternate between two activities: thinking and eating. To think, no (external) resource is needed. To eat, a philosopher must use both forks, the one on her left and the one on her right. An atomic action is requesting an adjacent fork, and, if available, taking it.[7] A philosopher who obtains both forks eats for a while and then releases the forks one after the other.

The original challenge calls for a *deadlock-free* program (in which at least one hungry philosopher gets to eat). We will also discuss a *starvation-free* program (in which all hungry philosophers get to eat). As the situation is symmetric, we demand that the concurrent algorithm be symmetric as well: namely, every

---

[4]This is of course a generality; each of these models has numerous submodels, and the relationship is more tricky; a comprehensive paper on these relationships is [AFR06].

[5]Written after most results described below.

[6]His paper [Dij65] on *mutual exclusion* marks perhaps the birth of the field.
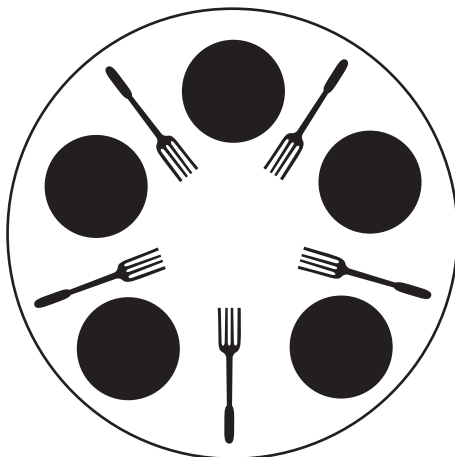
[7]So, access to a fork is *mutually exclusive.*

Figure 20. Dining philosophers' table.

philosopher has the same program. This requirement also implies complete symmetry of the philosophers themselves; they are identical, and in particular, have no unique names or IDs. A simple program P that illustrates the problem is the following.

1. *Try to obtain the left fork.*

2. *Try to obtain the right fork.*

3. *If unsuccessful in either attempt, go to 1.*

4. *Eat.*

5. *Release left fork.*

6. *Release right fork.*

It is clear that this program P is not deadlock-free. Imagine a scheduler that activates each philosopher for just one step (in any order). After this, each is holding her left fork, and this state will never change, regardless of the future activation order.

Of course, one can avoid deadlock: Simply consider the program that attempts to lift the left fork and then release it. The problem of course is that no one ever gets to eat. Can you find a starvation-free program? There were many suggested programs, but all went outside the model above and either broke symmetry (e.g., by giving the philosophers different IDs), or changed the problem (e.g., by making the philosophers "hygienic" [CM84], or by adding a "waiter"). But Dijkstra's original problem remained open, until Lehman and Rabin [LR81] proved that solving it is impossible.

Theorem 19.1 [LR81]. *There is no deadlock-free deterministic, symmetric program for the dining philosophers problem.*

A proof must present, for every such program, a scheduler that causes all philosophers to starve. The idea of the proof is simple (though formalizing it is not). The scheduler simply activates each philosopher for one step in a cyclical order, and repeats these rounds forever. The inductive claim, on the number of such rounds, is that before each round begins, the state of the system is symmetric (i.e., all philosophers are in the same state). This is true at the start, and the symmetry of the program ensures that, in each round, the result of the (inductively the same) step of any philosopher is the same. To conclude, assume that in some round, some philosopher is eating. The inductive claim and the fact that each philosopher executes exactly the same step with the same result per round, imply that, at the end of this round, all philosophers

are eating. But this is impossible, as each needs two forks to eat! Observe that asynchrony is crucial to this impossibility result, but it is used here in a rather benign way. The scheduler essentially prevents any player from executing two successive steps of its program before other players each execute one step in between.

But the main message of the paper [LR81] is that there *is* a solution if one allows probabilistic programs! This paper shows that a natural probabilistic variation of the program P above will foil the scheduling adversary, and some philosopher will get to eat with probability 1. Indeed, in the new probabilistic program R (which Lehman and Rabin call the "free philosophers algorithm"), essentially the only change is that when a philosopher who holds no forks tries getting one, she tosses a fair coin to decide whether to first try the left or the right fork. The correctness proof is quite nontrivial. It makes crucial use of a (natural) assumption: that every nonfaulty philosopher must be activated infinitely often by the scheduler. This assumption was removed by Duflot, Fribourg, and Picaronny [DFP04] (who also develop a very elegant formal framework to express the analysis of such probabilistic programs).

Theorem 19.2 [LR81, DFP04]. *There is a probabilistic symmetric program for the dining philosophers problem that is deadlock-free with probability 1.*

We conclude with another gem from [LR81]. These authors observe that, for program R, there are schedules that will prevent all but one of the philosophers from eating. While deadlock-free, this program is far from satisfactory (at least for hungry philosophers). They devise a new probabilistic program R' that fixes this problem and so is *starvation-free*! A key ingredient is allowing the state of a fork to tell us more than just whether it is in use or not. Recall that there was no restriction in letting a state carry more information, as long as it is finite and local. The new algorithm adds just one bit more to the state of a fork: which of the two philosophers adjacent to it ate last. (The initial state can be arbitrary.) Then program R', which they call the "courteous philosophers' algorithm," simply prevents the last eater adjacent to a fork from accessing it next. This modification guarantees that, in every schedule, every philosopher who is activated infinitely often will also get to eat infinitely often with probability 1!

Theorem 19.3 [LR81]. *There is a probabilistic symmetric program for the dining philosophers problem, which is starvation-free with probability 1.*

19.3   Coordination: Consensus and Byzantine generals

In the previous section, the problem we studied reflected a *conflicting* situation between participants, where they competed for shared resources, and the main problem was symmetry breaking. We now switch to a problem reflecting a *cooperative* situation, where players are jointly trying to compute a function (or perform a more general task) that depends on their private inputs, and where asynchrony is the main hurdle. We will consider it in both the *shared-memory* model and the *message-passing* model described earlier in the chapter. A common formal model for describing concurrent programs for such collaborative tasks, in both types of communication media, is the input-output automaton of Lynch and Tuttle [LT89] (but naturally, we will keep the discussion informal). The property of programs we will focus on, called *wait-freedom* in the shared-memory model (and sometimes simply *fault-tolerance* in the message-passing one), is that *every* "good" participant completes the task correctly in finite time, regardless of how all other participants behave. Here, "good" simply means "live" in the fail-stop fault model, and "honest" for the malicious (or Byzantine) fault model.

The problem we will focus on is the *consensus* (or agreement) problem. It came up very early for numerous practical applications in which occasional failure of components is possible. Among the most commonly mentioned applications are *transaction control* (e.g., two computers have to commit to when a transfer of funds from one bank to another was performed), *consistency control* (e.g., recovering from a crash, how a computer updates its memory from others that keep copies of the same information), *flight control* (e.g., air traffic control, where the waiting airplanes must agree on who gets permission to land), and *coordination* (e.g., a set of army generals has to agree on whether their units should attack or retreat).[8]

---

[8]The name "Byzantine generals problem", coined by [LSP82], is a common synonym for the consensus problem with potentially malicious faults.

Clearly, in all these (and other) examples, consensus is a kind of synchronization primitive that is absent (but needed) in the system.

More globally, considering the asynchronous, faulty environment the participants are in, it becomes evident that a consensus primitive would be extremely useful to have. For example, if they agree on a leader, then that leader could be the keeper of time and, hence, of the progress of computation; could possibly collect all other inputs and perform the computation himself; and so forth. Indeed, consensus feels like a natural "complete" problem. Of course, it is not that simple, as such a leader can fail, or worse—be malicious. But while not simple, it is true that the consensus task is complete, at least in the shared-memory model! A foundational paper of Herlihy [Her91],[9] which sets up the structural theory of wait-free programs, studies the relative power of *shared memory objects*, formally defines reductions between them, and proves completeness of consensus for such tasks. In summary, for intuitive, practical, and theoretical reasons, consensus is a basic problem—let us now define it and ponder its solvability.

The consensus problem   In words, "consensus" means that the players must all agree on *one* value, held initially by at least one of them. Let us state it a bit more formally, for the simplest case of binary values. Of course, some applications require large value domains, but our main concern here will be impossibility results.

Imagine n processors $P_1, P_2, \ldots, P_n$. Player P_i has a binary input value $x_i \in \{0, 1\}$. Its task is to commit to a binary output value y_i such that the following properties are satisfied:

- **Consistency** All committed values y_i are equal.

- **Validity** Every committed value y_i must be equal to some x_j.

Again, the challenge is to find a concurrent program in which every processor commits (or dies) in finite time. Note that the validity constraint prevents a trivial (and useless) solution in which they all commit to 0 regardless of their inputs.

The breakthrough paper of Fischer, Lynch, and Paterson [FLP85] shows that consensus is impossible in the message-passing model in the strongest imaginable sense.[10]

Theorem 19.4 [FLP85]. *For every $n \geq 2$, there is no program for consensus in the message-passing model that tolerates even one fail-stop fault.*

Moreover, the proof of this theorem goes through even if all messages sent are guaranteed to be delivered in finite time, and even if we require that only one live processor commits in finite time! Furthermore, despite the symmetry of the problem, the result holds not only for symmetric, identical programs (e.g., it holds even if each processor has and uses a unique ID).

I recommend reading the clear, detailed proof in the original paper [FLP85]. Here I highlight some high-level elements of the proof, as we will return to it later. The proof has two conceptual parts. First, it uses a hypothetical finite program to classify all configurations of the system that are reachable when running it. Then, it uses this classification to prove the existence of an infinite execution. Let us elaborate on these two parts.

To start, assume for contradiction that a given program has the property that on every execution,[11] *some* processor commits to a binary value in finite time. Take the first time that happens, and note that due to consistency, no other value can be later committed to by any other processor. This first value committed in each execution is now used to classify all *global configurations* (namely, the states of all processors and all message queues) that the program may arrive at as *univalent* or *bivalent*. We call a configuration *univalent* if all continuations yield the same committed value. We call it *bivalent* if some executions lead to a commitment

---

[9]Playing a similar role for distributed shared-memory wait-free algorithms, as the Cook-Karp-Levin papers on $\mathcal{NP}$-completeness (see Section 3.8) played for sequential polynomial-time algorithms and verification.

[10]Prior to this result, impossibility for 3 players and one *malicious* failure was proved in [LSP82]. An informal argument for impossibility in the special case $n = 2$ players was given in (the appendix of) [AEH75]. A good challenge for a motivated reader is to write down a formal proof for this simple case, if only to realize the need for precise definitions of what the processors and the scheduling adversary can do.

[11]Recall that each ordering of processor actions and messages delivered yields an execution, or run of the program.

of 0, and others to a commitment of 1. Clearly, no bivalent configuration is the final one. This classification sets up the following inductive argument for finding an infinite run. The (simple) base case is proving that there exists an input for which the configuration is bivalent. The (sophisticated) inductive argument shows that from *every* bivalent configuration, there is a finite schedule that leads to another bivalent configuration. This combinatorial argument, which carefully uses the properties of the model, is the heart of the proof. It now follows by induction that for some schedule, the program runs forever, proving that there is no wait-free program.

The notion of *bivalent*, and more generally, *indecisive* configurations (e.g., for nonbinary inputs) was extremely influential for future impossibility results. Indeed, Theorem 19.4 was extended to show the impossibility of consensus also in the shared-memory model, by Loui and Abu-Amara [LAA87] and by Herlihy [Her91].

Theorem 19.5 [LAA87, Her91]. *For every $n \geq 2$, there is no wait-free algorithm for consensus in the shared-memory model that tolerates even one fail-stop fault.*

Probabilistic programs　Of course, these results justify a variety of extra assumptions or relaxations of the problem used in practice, which actually lead to possible finite programs for this essential task of consensus. But like the case of the dining philosophers, the original problem is solvable if randomization is allowed.[12] Moreover, this positive result holds even under malicious faults.

The first probabilistic algorithm, for the message-passing model,[13] was discovered by Ben-Or [BO83].[14] It has two parts, depending on the type of fault; it can handle less than $n/2$ fail-stop faults, as well as less than $n/5$ Byzantine faults (the latter bound was improved by Bracha [Bra84] to $n/3$).

Theorem 19.6 [BO83, Bra84]. *For every n, there is a probabilistic concurrent program for the consensus problem that terminates in finite expected time under either of the following assumptions:*

- *less than $n/2$ of the players may fail;*

- *less than $n/3$ of the players are malicious.*

A key to Ben-Or's and all subsequent algorithms for this problem is a certain reduction of consensus to a *joint coin-flipping* problem. These original solutions, while finite, ran in expected exponential time, and the race was now on to improve their complexity. Most of this development, leading to some expected polynomial-time algorithms, is given in historical and technical detail in Aspnes' survey [Asp03]. But a central challenge has remained open since Ben-Or's original paper. His algorithm tolerates a "strong" adversary, namely, one who chooses which players to corrupt during (and depending on) the execution. As mentioned, the algorithm requires expected exponential time for a constant fraction of malicious faults, but if the number of bad guys drops below $\sqrt{n}$, then the complexity improves to expected polynomial time. The remaining challenge was the possibility of an expected polynomial-time program that can tolerate a constant fraction of malicious faults. This was finally resolved recently, 30 years after first being posed, by King and Saia [KS13] in the affirmative.

Theorem 19.7 [KS13]. *For every n, there is a probabilistic concurrent program for the consensus problem that terminates in expected polynomial time even if a (strong) adversary can corrupt up to $n/500$ players during the execution.*[15]

19.4　Renaming, k-set agreement, and beyond

We now return to deterministic programs, and the birth of the wonderful connection between topology and asynchronous distributed computing. Unless otherwise mentioned, all results stated are in the shared-memory model, which most subsequent work focused on.

---

[12]And "wait-freedom" is now defined as termination in finite *expected* time.

[13]And hence also for the shared-memory model, which can simulate it.

[14]The title of this paper refers to the Lehman-Rabin paper [LR81]. This is not surprising, as Ben-Or was Rabin's PhD student at the time.

[15]No attempt was made to optimize the constant—the proof is complex enough without it.

The impossibility of consensus, and attempts to understand the limits of the proof of Theorem 19.4, pushed researchers to define related problems to clarify the boundary between the possibility and impossibility of basic agreement tasks in asynchronous environments. One possible clue was the numeric coincidence: Consensus requires *one* common output value, and this is impossible with even *one* fault. The paper [ABND+87] studies some general *renaming* problems extending consensus, in which the set of possible output values must be smaller than the set of possible input values. In computer systems, shrinking the name space is a very common problem when many computational threads attempt to use a few common resources. The paper [ABND+87] gives some possibility and impossibility results that depend on the gap between the sizes of the two sets and on the number of faults allowed. However, an exact characterization of the boundary remained open.

To obtain a tight understanding after this paper, Chaudhuri [Cha93] defined a specific renaming problem, the *k-set agreement* problem, as follows.

Again we have n players, $P_1, P_2, \ldots, P_n$. Player P_i has an input value x_i from a set of size $k + 1$, say, $x_i \in \{0, 1, \ldots, k\}$. Its task is to commit to an output value y_i such that the following properties are satisfied:

- **Consistency**: The set of committed y_i has cardinality at most k.

- **Validity**: Every committed y_i must equal some x_j.

Again, the challenge is to find a concurrent program in which every processor commits (or dies) in a finite number of steps. Note that the consensus problem is a special case of k-set agreement with $k = 1$. Also note that the validity constraint remains the same.

The first (positive) result, obtained by Chaudhuri [Cha93], extends the obvious fact that consensus is possible with 0 faults and hints again at the same numeric coincidence.

Theorem 19.8 [Cha93]. *For every n and k, there is a wait-free protocol for k-set agreement tolerating less than k faults.*

To extend [FLP85] and prove impossibility when the number of faults is exactly k, Chaudhuri [Cha93] tried to imitate and generalize its proof technique to this case. She defined the natural analog of bivalent configurations, here *indecisive* (or $(k + 1)$-*valent*) configurations, which cannot be final (as there are still continued executions that can lead to all possible different commit values for the first committing players). It remained to give the analogous inductive proof that such indecisive configurations can be forced by a nasty schedule to persist forever. The paper proved only the base case of the induction, namely, that there exists an input that makes the initial configuration *indecisive*! Recall that this was very easy for $k = 1$. For $k > 1$, it is not, and a key contribution of this paper is the argument for proving the base case: using a tool we shall soon discuss called *Sperner's lemma*. The paper further expresses the hope that a similar argument can prove the inductive step, completing an impossibility proof.

The "Sperner lemma" hint was enough for three independent teams to quickly fulfill this hope. Borowsky and Gafni [BG93], Saks and Zaharoglou [SZ00], and Herlihy and Shavit [HS99] proved the impossibility of k-set agreement (even with k faults).

Theorem 19.9 [BG93, HS99, SZ00]. *For every n and k, there is no wait-free program for k-set agreement tolerating k faults.*

The proofs in the papers [SZ00, HS99] explicitly use *topology*. This connection has opened the door to the uses of a powerful mathematical theory for understanding asynchrony in distributed systems. In particular, it has the extremely appealing aspect of representing *dynamic* objects (like programs, schedules, and executions) as *geometric* or *topological* objects in a way that allows reasoning about the structure of static objects to reflect on properties of the dynamic ones. An excellent book describing these developments is [HKR13].

Theorem 19.9 was proved for the shared-memory model, and so, as discussed, also holds for the message-passing model. Note, however, that for that weaker model, a simpler impossibility proof was later discovered [BRS11] that is devoid of topology altogether; indeed, this proof uses a reduction in this model from consensus to k-set agreement, which allows the authors to directly apply the message-passing impossibility result Theorem 19.4.

Let us return now to the shared-memory model for the rest of the section. The three above mentioned papers proving Theorem 19.9 use topology to various degrees of explicitness.[16] While [BG93, SZ00] prove only Theorem 19.9, the proof in [HS99] derives this theorem from a much more general one that I will mention at the end, which uses *algebraic homology theory*. But first, I give a very high-level description of the proof—from the height chosen, they all look alike. However, I am mostly following the structure of the proof in [SZ00], and the mountain of details I am hiding under the rug here can be found in that paper, which also provides plenty of intuition and carefully explained special cases.

19.4.1 Sperner's lemma and Brouwer's fixed-point theorem

To begin with, let us state Sperner's lemma, which is a *combinatorial* statement. Then I will state the Brouwer fixed-point theorem, which is a *topological* statement. And then we'll discuss a reduction, showing how the second follows from the first. This will set up the stage for the proof of Theorem 19.9. The impossibility theorem for k-set agreement, which is a *computational* statement, reduces in essentially[17] the same way to Sperner's lemma just as the Brouwer fixed point theorem does! Indeed, the impossibility theorem may be viewed as a fixed-point theorem. We will restrict our attention to k-set agreement for $k = 2$, which means we'll need Sperner and Brouwer only for the Euclidean plane $\mathbb{R}^2$, which is easy to illustrate on paper. The same ideas carry over quite simply for general k, using the appropriate terminology of these mathematical results in $\mathbb{R}^k$, which we will avoid (e.g., replacing triangles by simplices, triangulations with simplicial subdivisions). You might want to imitate the proof below for the 1-dimensional case $k = 1$ (in which there are only line segments). In this case, Sperner's lemma and Brouwer's fixed-point theorems are really easy, but as noted, the consensus ($k = 1$) impossibility result is not.

We return to $\mathbb{R}^2$. We will need a few basic definitions. Consult Figure 21. Let T be a (geometric) triangle in the plane with vertices v_1, v_2, v_3. A *triangulation* D of T is simply a partition of T into a finite number of (smaller) triangles. The *vertices* of D, denoted $V(D)$, are all the vertices of these smaller triangles. A 3-coloring $\chi : V(D) \to \{1, 2, 3\}$ of the vertices of D is called *Sperner* if it colors the vertices of T with distinct colors, namely, $\chi(v_i) = i$, and colors every vertex u on the interval $[v_i, v_j]$ with one of its endpoint colors, namely, $\chi(u) \in \{i, j\}$. Finally, a triangle in D is called a *rainbow* triangle if its vertices have distinct colors.
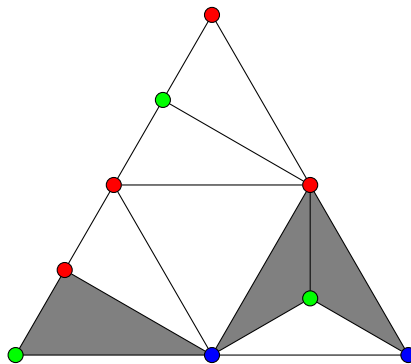
Figure 21. Triangulation and Sperner coloring. Rainbow triangles are shaded.

Clearly, if D is the empty triangulation (namely, it leaves T alone), then there is only one Sperner coloring, and it makes T a rainbow triangle. Once D is nontrivial (namely, vertices are added), T stops being a triangle in D, and there are several Sperner colorings. However, any way you do it, you cannot avoid a rainbow triangle. This is Sperner's lemma.

Theorem 19.10 [Spe28]. *For every triangulation D of T, and for every Sperner 3-coloring χ of $V(D)$, there is a rainbow triangle in D.*

---

[16]A later proof of Attiya and Castañeda [AC11] is completely combinatorial but clearly states the value of the topological viewpoint leading to their proof.

[17]A lot of hard work is hidden by this word.

The proof of this lemma is simple, essentially following from the fact that every undirected graph has an even number of odd-degree vertices (and if you've never seen this proof, please try proving it).

Here is Brouwer's fixed-point theorem, showing that another object is unavoidable: a fixed point in any continuous map from T to itself.[18]

Theorem 19.11 [Had10, BJ11]. *For every continuous function $f : T \to T$, there must be a point $x \in T$, such that $f(x) = x$.*

Let us derive this theorem from Sperner's lemma. Fix a continuous map $f : T \to T$. The reduction has three parts: First, obtain from f a 3-coloring χ_f of all points in T. Second, show that for *every* triangulation D, χ_f is a Sperner 3-coloring of $V(D)$. Third, apply this argument to finer and finer triangulations, and show that, in the limit, a rainbow triangle under χ_f is a fixed point of f. Let us first see how f induces a 3-coloring on *all* points in T.

1. χ_f is defined as follows. Let w be any point in T. Then, $w = \lambda_1 v_1 + \lambda_2 v_2 + \lambda_3 v_3$, with $\lambda_1 + \lambda_2 + \lambda_3 = 1$. Similarly, since $f(w) \in T$, $f(w) = \mu_1 v_1 + \mu_2 v_2 + \mu_3 v_3$, with $\mu_1 + \mu_2 + \mu_3 = 1$. Let $\chi_f(w) = i$ for the smallest $i \in \{1, 2, 3\}$ such that $\lambda_i \geq \mu_i > 0$. Note that such an i always exists.

2. Observe that every i satisfies $\chi_f(v_i) = i$ and that every $u \in [v_i, v_j]$ satisfies $\chi_f(u) = \min\{i, j\}$. Thus, in particular, for every triangulation, this coloring is Sperner.

3. Assume that $\{D_m\}$ is an infinite sequence of triangulations, where $D_0 = T$, and D_{m+1} a refinement of D_m such that every triangle of D_m is partitioned into triangles of area at most $1/2$ of the original area.[19] Clearly, $T = T_0$ is a rainbow triangle in D_0. By Sperner's lemma, if T_m is a rainbow triangle in D_m, there must be some rainbow triangle T_{m+1} in D_{m+1}. So, there is an infinite sequence of *nested* rainbow triangles. Let x be their intersection. By the definition of χ, x, and the rainbow property, if $x = \alpha_1 v_1 + \alpha_2 v_2 + \alpha_3 v_3$ and $f(x) = \beta_1 v_1 + \beta_2 v_2 + \beta_3 v_3$, we must have $\alpha_i \geq \beta_i$ for all i. But then $f(x) = x$.

19.4.2 Proof sketch of the impossibility theorem

With this proof in mind, we finally get to the impossibility theorem. Let us first recall the proof sketch of the impossibility Theorem 19.4 for consensus (and its adaptation to the shared-memory model Theorem 19.5, which is similar), and see how we can extend it to 2-set agreement. Given a hypothetically wait-free program π, the proof inductively generated an infinite sequence of *indecisive* configurations C_m of the system. First, it proved that some initial configuration C_0 is indecisive (recall that this had already been done for every k by Chaudhuri [Cha93]). Then, it proved that, if any configuration C_m is indecisive, there is a finite schedule that leads from C_m to another indecisive configuration, C_{m+1}. This is the hard part for $k > 1$ (which worked for $k = 1$). With plenty of hindsight, notice the syntactic similarity of the infinite sequence of *indecisive* configurations $\{C_m\}$, which are generated by a *single schedule* in that impossibility proof, and the infinite sequence of *rainbow* triangles T_m, which are *nested* in the reduction of the fixed-point theorem to Sperner's lemma. Ah, if we could only add the right semantics to this syntactic analogy! Let us try, now more carefully following [SZ00] (but more wildly oversimplifying, keeping the bare bones of its intuition). Again, we do it here for $k = 2$. And as suggested above, you may want to imitate this argument for $k = 2$ below with an argument for the simpler $k = 1$ (consensus) case, which may be revealing for some of the definitions.[20]

A key idea is to ignore the configurations for a while (whose description depends on the program we don't have) and instead focus on the schedules. There is really no loss in doing so, as fixing an initial configuration C_0 and a schedule (say, s), the program π determines the sequence of configurations it generates in the execution. Schedules are independent of the program, and their structure is simple. In the shared-memory

---

[18]Why is this a *topological* statement? Simply because the same statement holds for any subset of the plane that can be continuously deformed to a triangle. To see this, compose the deformation with the given function, find a fixed point, and invert the deformation.

[19]The refinement and area conditions on the sequence D_m are not necessary for this reduction—we do it to facilitate the next one.

[20]Both papers [SZ00, HS99] give a detailed account of the case $k = 1$ before starting the proof for $k > 1$.

model, they may be viewed as infinite sequences in $S = \Sigma^*$, where the alphabet Σ is all nonempty subsets of the players (namely, the subset of players that are allowed by the scheduler to *write* to memory in this step.[21]) S is a compact set, and we can imagine mapping it to the triangle T by a continuous map.[22]

The question is: How do we do that? The next key idea is that, in the reduction above, the sequence of nested triangulations was arbitrary, so let us pick it conveniently to accommodate such a map. Make each refinement look as in Figure 22, which may be viewed as the first triangulation[23], D_1. In it, every region is labeled by a nonempty subset of the three[24] players. Now imagine recursively triangulating every triangle in the same way (with a consistent labeling I will not describe, but which is natural). This yields the infinite sequence $\{D_m\}$.
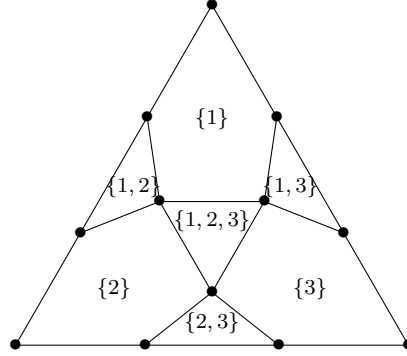
Figure 22. Basic triangulation, that is, D_1.

The nested sequence $\{D_m\}$ now supplies us with a natural canonical map μ of sequences S to T. If $s = \sigma_1, \sigma_2, \ldots$ is a sequence in S, it defines a sequence of triangles $t_1, t_2, \ldots$, with $t_m \in D_m$ in a natural way—if you are in t_m, then σ_{m+1} is the label of the next region t_{m+1} inside t_m. Let $\mu(s) \in T$ be defined as the intersection of all t_m. Now observe some important properties of this map μ.

1. The map μ is *onto*, so we have labeled the points of T by the elements of S.

2. Although the map μ is *not* a bijection, we will assume for simplicity that it is, namely, that μ^{-1} is well defined.

3. The map μ is *continuous*, under the natural metric on S: If two sequences s, s' agree on a prefix of length ℓ, then the distance between $\mu(s)$ and $\mu(s')$ is at most $|\Sigma|^{-\ell}$.

4. The map μ assigns a *face* (a vertex, edge, or triangle) in D_m to every *finite* schedule $s \in \Sigma^m$. There is a continuation in s in each triangle touching this face.

Now it is time to 3-color our triangulation and finally apply Sperner's lemma to conclude the proof. Indeed, let us define a 3-coloring χ_π of all points in T, using the program π, which we assume is wait-free. Take any $u \in T$, and let $s = \mu^{-1}(u)$ be the schedule mapped to u (which we assumed is well defined). By assumption, some player commits to a value under π on s in finite time; let this value be $\chi_\pi(u)$. This defines

---

[21]Reads also have to be incorporated, but we ignore this here.

[22]We ignore here a crucial point, which is elaborated in [SZ00], that does not affect our level of description. Namely, that one actually needs to define equivalence classes of schedules in Σ^* to define S, and this gives a different topology than the simple product topology on Σ^*. Roughly speaking, two schedules are *equivalent* if they induce the same sequence of contents of all shared registers.

[23]Imagine that the three pentagons in the "triangulation" are triangles, too. It is another detail swept under the rug, which is better omitted for the intuition. A simple fix is triangulating each pentagon by chords from the appropriate apex of the big triangle.

[24]Which suffices for proving impossibility when $k = 2$.

χ_π on all of T, and in particular, on all vertices of all triangulations. It is easy to see that this coloring is Sperner.[25]

Finally we can make sense of how configurations are modeled in this geometric picture, and define *indecisive* configurations. Consider any *finite* schedule $s = \sigma_1, \sigma_2, \ldots, \sigma_m$, and assume that $\mu(s)$ lies in the interior of a triangle T_m in D_m. Applying the program π to the initial configuration C_0 and schedule s arrives at some configuration C_m, which we can associate to this triangle. Define C_m to be indecisive iff this triangle is rainbow colored. The rationale is simple: By the definition of μ and the labeling scheme of triangulations, s can be extended in three different ways, so that each will reach a different vertex. Hence, there are three reachable configurations from C_m that reach three different colored vertices, namely, different committed values. This is precisely being indecisive!

Concluding, as we start from a rainbow triangle, $D_0 = T_0$, Sperner's lemma allows us to choose an infinite sequence of nested rainbow triangles T_m. Their intersection defines an infinite schedule, and the infinite sequence of configurations C_m defined by the program π starting with the initial configuration C_0 and following s are all indecisive! Hence, no processor can commit on this schedule s in finite time, contradicting the assumption that π is wait-free.

19.4.3   General input-output problems and simplicial homology

We continue to focus here on the shared memory model with fail-stop faults. As mentioned above, the paper of Herlihy and Shavit [HS99] does far more than prove impossibility for k-set agreement. Their paper considers all distributed tasks of this type, namely, all *input-output tasks*, an extremely rich set of problems roughly defined as follows. Intuitively, it allows any way of specifying which outputs are legal for each input, while including the possibility of participants failing.

More precisely, fix any alphabet Σ_I of input values and any alphabet Σ_O of output values. We assume that each contains a distinguished symbol $\perp$ (essentially denoting that a player can become inactive, or doesn't participate). Fix n to be the number of players. An *input-output task* is simply any relation $\Delta \subseteq I \times O$, where $I \subseteq \Sigma_I^n$ and $O \subseteq \Sigma_O^n$, which satisfies two natural closure properties: I and O are *monotone* sets under the $\perp$ symbol (namely, replacing the value in any component of a vector in the set by $\perp$ leaves it in the set), and Δ is *consistent* under $\perp$ (namely, for any $(i, o) \in \Delta$, if $i_k = \perp$, then also $o_k = \perp$).

Herlily and Shavit's main result is a precise *characterization* of the possibility or impossibility of a wait-free concurrent program for every such task. The characterization is in purely topological form, making no reference to protocols. This enables the use of topological tools to determine possibility for every task. Here is a simple example of this power: *A wait-free algorithm for 2-processor consensus exists if and only if there is a continuous onto map from a connected set in Euclidean space to another, disconnected set.* The latter is patently impossible, and hence so is the former.

More generally, the paper sets up natural topological spaces, called *simplicial complexes*, $\hat{I}$ corresponding to the possible inputs I, and $\hat{O}$ corresponding to the possible outputs O, so that every input/output vector corresponds to a simplex in the input/output complex. The characterization now becomes a question of the existence of a nice enough map between the two spaces. A bit more precisely (using undefined terms): *A wait-free algorithm for the distributed task Δ exists iff there is a simplicial map from a subdivision of the input complex to the output complex that respects Δ.*

I will not be more precise, nor will I define these terms here. But note that a simplicial map between complexes is an analog of a continuous map between topological spaces, and that the subdivision mentioned is an extension of the triangulation we discussed earlier. Indeed, the notion of a subdivision is what captures the protocol in this language, and we have seen this analogy in the proof of the impossibility of k-set agreement. In fact, when specializing this general set-up to the k-set agreement problem, the nonexistence of a simplicial map follows easily from Sperner's lemma (Theorem 19.10). In more general situations, tools from homology theory, which provide more sophisticated *invariants* of topological spaces (like higher dimensional notions of connectivity and "holes," homology groups, etc.) can be used to demonstrate the nonexistence of the

---

[25]A vertex v_i of T corresponds to a schedule where only player i ever gets to write, and so π must let it commit to its own value. Similarly, every point on the interval $[v_i, v_j]$ corresponds to schedules in which only these two players participate, and so the committed value must be one of theirs.

required map, as indeed is done in the original paper. As mentioned, further examples and extensions of the power of the topological method can be found in the book [HKR13].

19.5   Local synchronous coloring

This chapter ends with some positive news. There are plenty of impossible tasks in the asynchronous model that become possible when the players have a common clock (in particular, consensus and dining philosophers are such problems). We give another example, *local coloring*, which suits us well, as it brings together (a variant of) the renaming problem and computation in a ring of processors, both of which we have already discussed. But I had two much better reasons for choosing this gem, besides this convenience: *locality* and *determinism*. First, it demonstrates how surprisingly powerful and efficient a very loosely connected distributed system can be, despite the myopic view of individual processors limited by locality. Second, it demonstrates that, although randomness is an amazingly powerful way to break symmetry in distributed environments as we saw earlier, there are natural tasks of similar nature that can be solved deterministically.

Here is the set-up. Again we have a set of dining philosophers $\{P_i\}$ sitting around a table. Each can communicate only with its left and right neighbors. Each has an initial input value x_i from the set $[n]$, which is a legal coloring of the cycle (namely, each philosopher's initial value is different from those of its two neighbors). Note that there is no bound on the number of philosophers; indeed, it can be infinite! Despite this, the input values allow them to eat at a fixed, finite rate, with each philosopher sure to get a meal every n steps. How? Well, they have a common clock now, and they proceed in rounds modulo n. At time step t, every philosopher P_i with $x_i = t \mod n$ takes the two (available!) forks, eats for one time unit, and then puts the forks back on the table.

So far so good, but they are really hungry, and n can be huge, too huge a time to wait for the next meal (more seriously, it is clear that the rate of utilizing shared resources is important in numerous applications). Can one increase the rate? Sure! After all, every cycle can be legally 3-colored, so an external party knowing such a coloring could have given them values from $\{1, 2, 3\}$ that would allow each to eat every 3 steps. However, this is a distributed environment, and there is no such external party—they have to fend for themselves. How fast can a *local algorithm*, in which they exchange messages with their neighbors in each step, allow them to find such a legal 3-coloring starting from every possible n-coloring as above? This is the $n \to 3$ *coloring*[26] of the problem[27] at hand!

It is not even clear that such renaming can be found in any finite number of steps. Note that, in a T-step local algorithm (ignoring message sizes for now), every processor can find out everything about every other processor at distance T from it in the communication network connecting them (in this example, the cycle). So, the diameter of the network is an upper bound, and for many tasks, a lower bound as well (e.g., consider the 2-coloring problem of an (even) cycle [Lin92]). But here we have no bound on the size of the network (which for a cycle is also essentially the diameter)! So, a finite algorithm will have to depend on the values in a small neighborhood.

A natural idea is indeed to use randomness: Let each P_i choose a random color from $\{1, 2, 3\}$ and check its neighbors' values. Only a constant fraction (at most 5/9 in expectation) of processors will be in conflict (i.e., have a neighbor with the same color). Now, those without a conflict stay put, with their choice of color fixed forever. Each processor in conflict tosses again (from among its current color and the colors not chosen by either of its neighbors, leaving it at least 2 possible colors). Again, we expect only a constant fraction of them to still have a conflict, so they can repeat until there is no conflict. This idea seems to beat the diameter bound: If there are N processors on the cycle, then we expect termination in about $\log N$ steps, in expectation (and even in fact) with high probability. But this algorithm has several shortcomings. First, N might be huge. Second, they have no obvious way to know that no conflict remains, namely, a time by which the philosophers know the process has ended and they can start eating. Finally, they did not use their inputs—can these help?

---

[26]It is a renaming problem (as the processors choose different names) from a smaller palette, but now the consistency requirement is of a different nature.

[27]One can, and we will, consider $n \to m$ coloring for any $m < n$.

An ingenious local algorithm suggested by Cole and Vishkin [CV86] shows how they can do far better, *deterministically*. Indeed, the number of steps needed is very nearly a constant!

Theorem 19.12 [CV86]. *There is a deterministic local algorithm for the $n \to 3$ synchronous renaming problem that takes $O(\log^* n)$ steps.*[28]

The idea is simple. Let's see how processors can shrink the set of colors from n to $2(1 + \lceil \log n \rceil)$ in one step. Think of names as k-bit sequences for the smallest k such that $n \leq 2^k$. Consider any processor P with input value x, whose (different from it) neighboring values are y on the left and z on the right. Let i be the leftmost bit for which $x_i \neq y_i$ and j the leftmost bit for which $x_j \neq z_j$. Now this information suffices for every processor to create a much shorter value, maintaining the property that these values form a legal coloring. The new value P chooses is x', simply defined as $(i, x_i), (j, x_j)$, which is of length $2(1 + k)$, as needed. I leave to you the (slightly subtle) task of verifying that this is a legal coloring. Now, iterating this process $2 \log^* n$ steps results in some C-coloring with some $C < 20$. It is another exercise to see how in C more steps, one can further reduce it to a 3-coloring.

This result is important for the applications discussed above, as well as being a general technique to achieve some kind of local (enhancement of) symmetry breaking, deterministically, in a number of steps that is nearly independent of the size of the network and its diameter.

Several natural questions which arose given this work were addressed by Linial [Lin92]. First, can one do better than the algorithm above, perhaps solving this coloring problem in a constant number of steps? Second, can one handle other communication networks besides the cycle? Third, can randomness be used to improve any of these algorithms? We conclude with Linial's answers (no, yes, and no, respectively):

Theorem 19.13 [Lin92].

- *Any local algorithm for the $n \to 3$ coloring problem on the cycle requires at least $\frac{1}{2} \log^* n$ steps.*

- *There is a local algorithm solving the $n \to O(d^2)$ coloring problem on any graph of maximum degree d in $O(\log^* n)$ steps.*[29]

- *If a probabilistic local algorithm solves a coloring problem in T steps in expectation, then there is a deterministic T-step local coloring algorithm for the same problem.*

---

[28] All logarithms are base 2. Recall that the $\log^*$ function is the inverse of the tower function, namely, $\log^* n$ is the smallest number h such that a tower of 2's of height h exceeds n, or equivalently, the smallest number of times h of logarithms we need to take, starting at n, to get below 2. This function grows extremely slowly. For example, if n is the number of atoms in the universe, estimated by 10^{80}, then $\log^* n = 4$.

[29] And thus, also an algorithm that in $O(d^2 + \log^* n)$ steps solves the $n \to d + 1$ coloring problem.

20 Epilogue: A broader perspective of ToC

This book has focused mostly on computational complexity, both as a major mathematical area, describing its rich internal structure and its broad reaches into other areas of mathematics, and as a central, pivotal area of the theory of computation (ToC). This chapter is devoted to a panoramic view of ToC that extends far beyond the intersection of math and computer science. The great impact of technology on society tends to obscure the intellectual foundations that ToC rests on from laypersons, but sometimes from experts as well. Even to many who know and understand them, these theoretical ideas are often viewed only as servants of the technological development. In this chapter, I elaborate on the nature of ToC as an independent intellectual discipline, which is summarized as follows.

The theory of computation, since its inception by Turing in 1936, is as revolutionary, fundamental, and beautiful as the great theories of mathematics, physics, biology, economics... that are regularly hailed as such. Its impact has been similarly staggering. The mysteries still baffling ToC are as challenging as those left open in other fields. Moreover, the ubiquity of computation makes its theory central to all other disciplines.

In creating the theoretical foundations of computing systems, ToC has already played, and continues to play, a major part in one of the greatest scientific and technological revolutions in human history. But the intrinsic study of computation transcends human-made artifacts, and underlies natural and artificial processes of all types. Its expanding connections and interactions with all sciences, integrating computational modeling, algorithms, and complexity into theories of nature and society, is at the heart of a new scientific revolution!

The sections in this chapter discuss at a high level various aspects of ToC, mainly intellectual but also social and educational. This exposition aims at describing the nature and scope of the field, its evolution so far, and the important role I expect it to play across the intellectual and societal arenas. I hope it will give a useful bird's-eye view of the field to newcomers and motivated outsiders, as well as to computer scientists. This exposition is clearly biased by my personal experience, views, and limited knowledge.

Organization The first several sections discuss the connections and interactions between ToC and many other disciplines. I first discuss, rather briefly, its interaction with its "parent" fields, computer science and engineering (CS&E) and math. I then move on to describe in more detail interactions with "neighboring" fields, like optimization and coding theory. Finally, I discuss at length interactions with more "remote" sciences, like biology and economics, as well as with philosophy, and with technology.

In between the sections about neighboring and remote disciplines we will take two important detours. One will discuss a very *general definition of computation*, which will clarify the broad mission of ToC and how it is naturally aligned with the basic missions of all sciences. The second will give a general description of the *central tenets of ToC methodology*, which will clarify what new aspects ToC can bring to these joint collaborations with the sciences.

After discussing connections and interactions, I will turn to some high-level challenges of ToC, and then briefly discuss the role and importance of ToC research and the impact researchers can have on K–12 education. I will conclude with some personal observations on the socio-academic character of the field, which I believe partly explains its remarkable success so far and should be preserved for future success.

Taking stock of a scientific field's history, evolution, ideas, contributions, and challenges is very familiar in fields that have existed for centuries. Numerous semi-popular books have been written on mathematics, physics, biology, and other fields from this viewpoint. ToC, counting from Turing's paper, has existed for 80 years (of which I have personally witnessed the last 40). I believe that there is clearly room for a book on ToC that will describe and argue in more detail its achievements, goals, and independent intellectual place in the academic sphere. Moreover, I believe that such education of members of the field, academia, and the public will be important for the field's future development. Here, I have condensed my version of such a treatment of the field into one chapter.[1] Besides being short, this survey is only a time capsule. I fully

---

[1]Twenty years ago, Oded Goldreich and I gave a similar treatment [GW96] of the nature and future of the field, with the same conclusions as to its success, promise, and independence, but one that was much shorter and had far less detail. The two decades that have passed, and all the additional research ToC has created since, make it easy for me to add much more supporting evidence for these claims.

expect that the enormous expansion rate of the role computation plays in every aspect of our lives will likely change and expand the scope, goals, and mission of ToC. As computation is ubiquitous, the potential for such expansion is unlimited.

Throughout this chapter, "ToC research" will refer to the pursuit of understanding computation in all its forms rather than to the disciplinary affiliation of the researchers who perform it; although these researchers are typically affiliated with ToC, notable contributions of this nature have also been made by applied computer scientists, mathematicians, and scientists of all disciplines.

20.1   Close collaborations and interactions

It is hard to do justice in a few pages to the breadth and variety of interactions of ToC with its "nearby" disciplines, all much larger than it. I do not attempt to fully describe this immense body of work. Instead, I try to capture the different nature and role these interactions take from each different discipline, and how in turn they shape and reshape ToC and its reach. I start from the closest disciplines, math and CS&E (computer science and engineering), and then continue into neighboring fields. Both here, and when we turn to interactions with more distant fields, the coverage and level of detail are not uniform.

Meet the parents: CS&E vs. mathematics   As discussed in Chapter 1, ToC was born out of, and lived within, two extremely different traditions, which shaped its unique character. I consider this combination of genes and environments extremely fortunate! Here are some examples. CS&E is a young and impatient field, whereas math is old and patient. Research directions in CS&E are mostly driven by practice and utility, while in math they are mostly driven by æsthetics and abstraction. CS&E demands the study of real, functioning computer systems, while math demands any study to be fully rigorous but allows the imagination to run free without regard to reality. Each alone and both together generate an unending host of modeling and algorithmic challenges for ToC. I will be rather brief on both.

20.1.1   Computer science and engineering

The immense technological achievements of the computer industry that surround us blind many, not only laypersons and politicians but scientists as well, to the pure intellectual wonders that brought many of them about. ToC, besides creating the many theories underlying these technologies, continues to build the conceptual infrastructure for all computer professionals and the problems they address. Most fundamentally, this includes the ability to precisely model diverse computational settings, to formally define resources, and to analyze their utilization when organizing and processing information by algorithms. ToC has provided novel, useful definitions and methods to study users' access to knowledge (and thus can assess privacy and secrecy); the power and limits of randomness and interaction; the organization, utilization, and analysis of systems with many components, and many other computational settings, some covered in detail in this book. These conceptual contributions, some of revolutionary practical consequences, have become ways of thought so ingrained in basic undergraduate education that they are often taken for granted.

The specific contributions of theory to CS&E activities are very well documented, so I will be telegraphic here, confining myself to name dropping. ToC has created and is still developing foundational theories that underlie much of the development of computational systems of all kinds. In many cases, theory predated practice and enabled system development.[2] In many cases, theoretical ideas born and developed for a particular technology remained useful after that technology became obsolete.[3] Here is a partial list of theories, each member of which is deep, broad, and occupies volumes of textbooks (which one should study to fully

---

[2]Examples abound, and I give only two primary ones. The first is the early theoretical blueprints of the computing machines of Turing and von Neumann, which unleashed the digital age. The second is the theoretical foundations of public-key cryptography, which enabled potential e-commerce applications, that in turn unleashed the rapid expansion of the Internet.

[3]Again, examples abound, and I give two representatives of two phenomena. First, theories: The theory of communication complexity, was designed primarily to understand area-time trade-offs in VLSI chips and has gone on to be key in numerous other areas, as we discussed in Chapter 15.2. Second, algorithms: The algorithm for pattern matching, designed initially for text processing, was (together with its extensions) essential for the Genome project.

appreciate the underlying theoretical foundations). It starts with Turing's definition of computation and algorithms, and proceeds with theories of cellular automata, finite and infinite automata, programming languages, system verification, databases, computational complexity, data structures, combinatorial algorithms, cryptography, distributed systems, software engineering, randomness and pseudo-randomness, computational learning, approximation algorithms, parallel computation, networks, quantum computation, online and real-time algorithms, and many more. Needless to say, the rapidly evolving world of computer science and technology continuously supplies many varied situations to model as well as problems to solve, such as the recent sublinear algorithms for huge datasets; differential privacy for scientific and public databases; delegation in cloud computing; population protocols in sensor networks; and of course, the struggle for theoretical explanations of machine learning programs, like deep networks. Future technologies and applications will undoubtedly lead to new theories, more interaction, and better understanding of the nature, applicability, power, and limits of computation.

20.1.2 Mathematics

With math, the nature of interactions developed differently. Again I will be brief, as this book has many examples throughout, with the Interlude Chapter 13 specifically devoted to a select few of them. Early on, aside from the early intimate collaborations with logic, ToC mostly used mathematical techniques and results. Dealing mainly with discrete objects, it was natural that most initial interactions were with combinatorics. But as ToC broadened and deepened, its varied theories required tools from diverse mathematical fields, including topology, geometry, algebra, analysis, number theory, and algebraic geometry, among others. Such tools were often not readily available, or did not even exist, and so had to be developed. This formed many collaborations, which led to purely mathematical results in all these areas. Another source of new questions (and the need for further tools) comes from completely new mathematical models for different computational phenomena, as some of the chapters of this book illustrate. These activities were combined with the meta-challenge of making math algorithmic; namely, *efficiently* finding objects whose existence was proved by some indirect or nonexplicit arguments. Such questions have pervaded mathematics, in some cases for centuries. Pursuing them, especially with the newly developed algorithmic and complexity tools, has led to rethinking and expanding many areas, discovering new structures, and raising many new questions. These rapidly expanding collaborations with many areas of mathematics have greatly enriched ToC as well and clarified its own independent role as an important mathematical area. It is heartwarming to see how most young mathematicians are well versed in the basic notions, results, and major open problems of computational complexity, as they would be about any other math area outside their speciality. I have no doubt that the algorithmic and complexity ways of thinking will further penetrate all areas of math and will create many new interactions between other fields.

Meet some neighbors: Optimization, coding and information theory, statistical physics   I now give some more details and references on the many and growing interactions between ToC and three very large disciplines that are naturally quite close to computer science: optimization, coding and information theory, and statisical physics. These connections are already surprisingly diverse and in some cases have surprising origins. I find this array of connections stunning in its breadth and depth. Again, the topic list and references given here are partial.

20.1.3 Optimization

The connections of the large optimization community with ToC is far from surprising, as efficient algorithms are a core object studied by both (even though initially the types of problems and tools each community focused on were somewhat different). However, what may not have been expected is how major breakthroughs, which mostly arose from *computational complexity* considerations and constructs, would enrich and rejuvenate the study of the power and limits of algorithms. Many of these are discussed in this book and include:

- The PCP theorem, leading to a theory of hardness of approximation, analogous to (though far more difficult and refined than) the theory of $\mathcal{NP}$-completeness. Landmarks include [FGL$^+$96, AS98, ALM$^+$98, Kho02, Rag08] and some of these developments are discussed in Sections 4.3 and 10.3. It is interesting to note that the origins and methods leading to the PCP theorem are far from optimization: They include cryptography, interactive proofs, coding theory, program testing, and average-case complexity.

- The power, limits, and connections of LP and SDP hierarchies, powerful algorithmic paradigms existing in the optimization literature, which were (and are) becoming much clearer with works like [Gri01b, Gri01a, ABL02, BS14, LRS14, CLRS16]. Some of the origins and connections leading to these developments include proof complexity, computational learning theory, and random restrictions based on circuit complexity.

- Related and more specific to the item above is the "extension" of linear programs, namely, adding variables to reduce the number of inequalities. The exact power of this technique was fully characterized in [Yan91], and its limits were first determined in [FMP$^+$15] (and then further developed in subsequent work, also mentioned in the previous item). Interestingly and surprisingly, some of the crucial insights and methods came from seemingly unrelated areas, including communication complexity and quantum information theory.

- Continuous optimization methods starting perhaps with Karmarkar's algorithm [Kar84] for linear programming. More recently, momentum has picked up considerably, and the ellipsoid method, interior point method, alternate minimization, multiplicative weights, and a variety of first and second order descent methods (as well as their applications), have been greatly extended: [ST04a, CKM$^+$11, AHK12, Mad13, LS14, KLOS14, AZO14, GGOW15, AZH16]. In many cases, these methods have broken efficiency records held for decades, and some resulted in near linear algorithms. Connections and tools range from electrical flows and random walks to permanent approximation, spectral decompositions, and functional analysis.

- The exponential time hypothesis (ETH). This hypothesis has been proposed as a natural but much stronger hardness assumption than $\mathcal{P} \neq \mathcal{NP}$ in [IPZ01, IP01]. The ensuing active theory, often called *fine-grained complexity* (as it introduces more delicate notions of reductions between optimization problems) enables predicting the precise complexity of problems in the class $\mathcal{P}$, and in particular ruling out linear time algorithms (see, e.g., the survey [Wil15]).

- Smoothed analysis. Introduced in [ST04b], it provides a radically new way of analyzing heuristics, very different from all previous average-case models, and serves to explain their success on "typical" inputs in a new way.

20.1.4   Coding and information theory

The connections of ToC with the vast field of coding and information theory are extremely broad, ranging from the ordinary to the very unexpected. Most expected connections have to do with the fact that computations (in computers, networks of computers, databases, etc.) process information that is subject to noise and error, and so must be designed to be fault tolerant. This naturally begs the use of error-correcting codes and information theory. The less-expected connections include the introduction of *locality* into error-correcting codes (in several ways), the rejuvenation of several old ideas and models (like list-decoding, low-density parity-check codes, and the role of interaction in coding and information theory), with new questions, applications, techniques, and results. I list and reference some of these below (many are discussed throughout the book, and especially in parts discussing interaction, especially Section 15.3 and Chapter 10).

- The idea of list-decoding (namely, that codes can tolerate far more noise if decoding is relaxed to produce a short list of potential messages, as opposed to a unique one) goes back to the very early days of coding theory [Eli57]. Forty years later, starting with the breakthrough work of Sudan [Sud97], a sequence of works including [GS98, PV05, GR08b] quickly led to optimal list–decodable codes with efficient encoding and decoding algorithms, and the study of list decoding for many different codes.

- In contrast, the idea of studying *local decoding* arose from complexity-theoretic considerations in cryptography, de-randomization, and PCPs. In local decoding, only one (specified) bit of the original message should be recovered from a noisy encoding of it, but only a tiny (random) fraction of it can be inspected, and small errors are allowed. Remarkable constructions and applications appear in [GL89, STV99, KS09, Yek12, Efr12, DSW14b, DG16] and in many other papers. As one example of an application, the related *locally repairable codes* of [GHSY12] and its followers had a tremendous effect on the design of distributed data centers of Internet data, where recovery from faults and consistency across multiple copies became a completely new challenge at these volumes and speeds. Some of the many complexity-theoretic applications of local decoding (often combined with list decoding) can be found in the survey [Sud00], and many others, in algebraic complexity, combinatorial geometry and information retrieval appear in some of the literature.

- *Local testing* of codes is another new coding problem that was born from computational complexity considerations of program testing, interactive proofs, and PCPs. Here one simply asks whether a given word is in a given code or very far from it (in Hamming distance), again by inspecting only a (random) tiny part of it. Once more, remarkable constructions and applications can be found, for example, in [BFL91, BLR93, GS00, RS97, AS03, GS06, IKW12, KMRZS17, DK17].

- In the reverse direction, the idea of *concatenated codes* [For66], developed in coding theory to construct explicit, efficient codes, inspired many of the proof composition constructions of PCPs and related systems. Other notions of robustness as well as actual constructions were borrowed, directly and indirectly, from the recovery from errors in robust codes to recovery from errors in robust proofs.

- "Graph-based" or LDPC (low-density parity-check) codes were first proposed and studied in the seminal work [Gal62]. However, this direction was left mostly dormant until a surge of interest and results brought it back 30 years later, much of it due to the rise of *expanders* (discussed in Section 8.7), graphs with diverse applications in ToC. These initial works include [Spi95, SS96, LMSS01, RU01, Lub02, CRVW02].

- Interactive computation was always part of information theory. Nevertheless, communication/information complexity, and interactive coding theory have emerged and have grown into two well-developed theories *within* computational complexity, with major applications in the field. These theories have considerably enriched the interactions with coding and information theory. Both are described in Chapter 15.

20.1.5 Statistical physics

It may seem surprising that I include statistical physics as a neighboring field to ToC, but many natural connections and common interests between the two were discovered very early on and have led to significant collaborations (that naturally included discrete probability and combinatorics). Moreover, many of the problems and methods of statistical physics are naturally algorithmic, as we shall presently see. I will briefly survey this common ground, so as to better appreciate the new interactions (making this section somewhat longer than the previous two).

A central theme of statistical physics is understanding how global properties of a large system arise from local interactions between its parts. A commonly given example is the *global* states of matter: gas, liquid, solid, and the transitions between them (e.g. turning ice into water and then into vapor) under temperature change, which affects the movement of molecules and the *local* interactions between neighboring ones. Another common example is magnetization under the influence of an electric field. A system is typically described by a graph, whose nodes represent the interacting parts (each of which can be in some finite number of states) and whose edges represent the local interaction structure[4]. These give every global

---

[4]This (discrete) formalism captures a large number of models of matter of all types, including spin systems on glassy and granular matter, electromagnetic systems (both classical and quantum), "hard-core" interactions, and percolation in porous materials. Many other variants we will not discuss include continuous settings like heat baths, Brownian motion and other diffusive systems, and nonlinear interactions, for example, the Maxwell-Boltzmann ideal gas model (which gave birth to statistical physics), and billiard models.

state a weight, or *energy*, naturally inducing a probability distribution on global states of the system called the *Gibbs distribution*. Determining global properties of the system (mean energy, long-range correlations, phase transitions and others, all with physical meanings) is generally reduced to *sampling* a state of the system according to the Gibbs distribution. This sampling problem is a natural computational problem! Note, however, that the number of states is exponential in the size of the system (the number of parts), so it is nontrivial.

It is not hard to see that this discrete formalism naturally captures and indeed generalizes *constraint satisfaction problems* (CSPs) like $3-SAT$, which I discuss in Section 4.3. Descriptions of such systems (the interaction graph and the local energy functions) comprise the input data to many optimization problems of algorithmic interest. But typically, one asks not to sample a random state, but rather to find one that (exactly or approximately) optimizes the energy. Similarly, in combinatorics, the same data are considered as input to many enumeration problems, that is, counting the number of states with optimal (or specific) energy. It turns out that all these computational problems, *sample, search, count*, are related, which naturally fosters interaction, which we will discuss.

A sampling method originating with von Neumann and Ulam in the 1950s, developed partly for the Manhattan project, is the so-called *Monte Carlo* algorithm, which was subsequently further developed. In the framework described above, one sets up an exponentially large, implicitly described *Markov chain* on the states of a system, whose stationary distribution is the Gibbs distribution (this is called the *MCMC method*, for Markov chain Monte Carlo). Then, starting from an arbitrary state, one proceeds with the natural random walk on the chain in the hope of converging quickly to a typical state, as required. Natural algorithms of this type include the Metropolis algorithm and Glauber dynamics. Numerous simulations of such chains were and are carried out to determine properties of numerous models. However, few tools existed, for precious few systems, that could rigorously quantify the convergence rate of these algorithms, and so heuristic arguments (and resource limitations) were used to cut off the simulation time. Needless to say, if the random walk did not converge, the information deduced about the global state of the system could be completely wrong. And, of course, MCMC may not be the only way to sample! Interaction with ToC started when the field began investigating *probabilistic algorithms* in the 1970s, of which MCMC is a perfect example. This has led to significant progress on the sets of problems described above regarding local interacting systems, which I briefly summarize:

- Valiant's complexity theory of counting problems [Val79b, Val79c] introduces the complexity class $\#\mathcal{P}$ and establishes that the permanent of Boolean matrices is complete for this class. This allows us to demonstrate hardness of almost all natural enumeration problems above, and with them, the hardness of computing the probabilities in the associated Gibbs sampling problems (indeed, dichotomies distinguishing hard and easy counting were established—see the comprehensive book [CC17]).

- To sweeten the pill, Jerrum, Valiant, and Vazirani [JVV86] prove that *sampling* is equivalent, for most problems of interest, to *approximate counting*. This provided a key connection that makes sampling algorithms applicable to solving enumeration problems.

- A series of papers by Jerrum and Sinclair [JS89, SJ89] provided general methods of *canonical paths* and *conductance* to bound the convergence of general Markov chains, which these authors used to rigorously prove polynomial convergence for problems in statistical physics like the Ising and monomer-dimer models, and enumeration like counting matchings in graphs. These papers had a flood of follow-ups, developing these and other methods, including rigorous ways of using various forms of coupling. Many early examples are summarized in [Jer96].

- The important work of Jerrum, Sinclair, and Vigoda [JSV04] gave a polynomial time probabilistic algorithm to approximate the permanent of nonnegative matrices, which by completeness (above) captures many other enumerations problems. Efficient *deterministic* algorithms for the permanent, albeit with only an exponential-factor precision [LSW00, GS14], exposed the connections of these problems to matrix scaling and hyperbolic polynomials. These aspects and many more are explained in Barvinok's book on combinatorics and the complexity of partition functions [Bar16].

- The connection between *spatial*, structural properties of local systems (e.g., long-range correlations in the Gibbs distribution, phase transitions) and *temporal*, complexity theoretic properties (e.g., convergence time of natural Markov chains like Glauber dynamics) has been studied by physicists in spin systems since the 1970s. This connection was expanded by the work of Weitz [Wei06] on the *hard-core* model; he developed a *deterministic* algorithm (very different from the Markov chain approach) that is efficient up to the phase transition. This was complemented with a a hardness result of Sly [Sly10] just above the phase transition. This triggered further collaboration and a better understanding of this deep relationship between spatial and temporal mixing (see [DSVW04] for a survey).

- The *Lovász local lemma* (LLL) enables us to establish the *existence* of rare "global" events. Efficient algorithmic versions of the LLL were initiated by Beck [Bec91] and, starting with the work of Moser [Mos09] (and then [MT10]), have led to approximate counting and uniform sampling versions for rare events (see, e.g., [GJL16]). These techniques for analyzing *directed, nonreversible* Markov chains are a powerful new tool for many more applications. A completely different deterministic algorithm of Moitra [Moi16] in the LLL regime promises many more applications; it works even when the solution space (and hence the natural Markov chain) is not connected.

- Finally, Markov chains like the Metropolis algorithm, guided by an optimization objective like energy, have been used as optimization tools in heuristics, such as *simulated annealing*, to generate Gibbs distributions that favor states of high objective value. Some of these techniques allow analysis of convergence time for classical specific optimization problems (see, e.g., [JS93] for upper bounds and [Jer92] for lower bounds).

20.2   What is computation?

As we now move away from interactions of ToC with neighboring fields to (seemingly) more remote ones, we take a higher view. It is fitting at this point that I should explain first, as we discuss the theory of computation, what we mean by the term *computation*. One of the broadest ways to informally define computation — indeed, the view that underpins the celebrated *Church-Turing thesis* (which is discussed more later), is as follows.

> *Computation is the evolution process of some environment,*
> *by a sequence of "simple, local" steps.*

If this definition seems to you as capturing practically any natural process you know of, that's precisely how I want it to sound!

Of course, this definition calls for a specification of the evolving environment and a notion of granularity that will distinguish local and global, simple and complex. The most basic setting (from which the word *computation* originally arises), is when *bits* evolve in a Turing machine or in a Boolean circuit; this is one concrete example of the granularity notion, and the rules/steps of evolution. Another, still an actual computation but with completely different granularity and basic steps, arises in the evolution of the *states* of processors in a network, under (say) pairwise communication. And there are of course many other examples which capture (existing or imagined) computing systems.

The main point I wish to make is that this viewpoint of processes as computation extends to numerous other settings, vastly removed from computers. In each setting, many different choices of granularity, and simple and local rules (which may be given by nature or made up by us), will lead to different evolution processes. All of these are worth studying (even when physical) as *information processes* from a computational perspective, using the methodology of ToC that we elaborate on in Section 20.3.

Here is a partial list of environments with such interacting parts, which in all cases can shed their physical characteristics and be viewed as transformations of pure information:

- Bits in a computer.

- Computers in a network.

- Atoms in matter.

- Neurons in the brain.

- Proteins in a cell.

- Cells in a tissue.

- Bacteria in a Petri dish.

- Deductions in proof systems.

- Prices in a market.

- Individuals in a population.

- Stars in galaxies.

- Friends on Facebook.

- Qubits in entangled states.

All of these are playgrounds where theorists of computation[5] have a role to play! The computational modeling and quantitative study of the resources expended by all such processes, what they "compute" and other properties of this evolution, is the bread and butter of ToC.

These and many other similar examples clarify that the notion of computation far transcends its (natural and essential) relevance to computer technology, and demonstrate the need for a theory of computing, even if computers did not exist at all!

20.3   ToC methodology

As we have seen, many natural and human-made phenomena present us with processes we would like to understand, and the many problems of practical and intellectual importance present us with the need to develop efficient ways to solve them. ToC has created a powerful methodology and language with which to investigate questions of this type. Below are listed some of its (interrelated) important principles, which we have seen in action repeatedly in the previous chapters of the book, and should be considered in the general context of computation we have discussed in this chapter. Let me stress that most of these principles are not new: They have been in use in many studies across science and mathematics for ages. I feel, however, that they are handled in a more systematic way in the context of ToC, and in a sense, some of these principles themselves become objects of study of the field. I expect that their more systematic use in math and other sciences would be rewarding, as existing interactions (some of which are discussed in the following sections) already reveal.

1. **Computational modeling**: *Uncover and formally articulate the underlying basic operations, infor-mation flow, and resources of given processes*. This book has many examples of computational processes with a large variety of basic operations, like Boolean or arithmetic gates, which can be deterministic, randomized, or quantum. We have seen, for example, geometric, algebraic, and logical deductions in proofs. In all of them, we have studied time, space, communication, randomness, and other resources. Beyond computer science and math lie a vast number of natural processes using different resources, many of which can be viewed as information processes. Modeling their basic steps and resources *as* computation, and applying the computational language, methodology, and results may be extremely beneficial in the sciences. Moreover, the abstract, computational understanding of natural processes may feed back into computer technology by integrating algorithms and hardware used by nature, as ini-tial attempts in nanocomputing, quantum computing, DNA computing, swarm computing, and others promise.

---

[5]And these theorists can come from every discipline.

2. **Algorithmic efficiency**: *Attempt to minimize relevant resources used by computational processes and study their trade-offs.* It should be stressed at the outset that this principle applies equally to algorithms designed by a human to solve a problem, by deep networks trained on massive data to self-improve, or by algorithms that evolved in nature over eons to guide the behavior of living organisms. In all, economizing resources is primary (even inanimate physical objects seem to prefer low energy states). Experience shows that studying the limits and trade-offs of efficiency is a great classification guide for problems and processes. Moreover, developing general analytic tools to find the limits and trade-offs of efficiency in one computational setting can be extremely powerful in others.

3. **Asymptotic thinking**: *Study problems on larger and larger objects, as structure often reveals itself in the limit.* There is a natural, practical tendency to focus on understanding (or developing algorithms for) concrete, specific objects that we really care about, which have specific sizes (e.g., the human genome, the Facebook graph, the roadmap of the United States, Rubik's cube, the proof of Fermat's last theorem). However, viewing such concrete objects as parts of infinite families, to which the same process or algorithm applies, may lead to more efficient and more general methods of treating even these particular objects. Some of the major successes of the field stem from applying the asymptotic viewpoint, and numerous algorithms, reductions, and complexity classes demonstrate this clearly. This approach has parallels in other fields. In coding theory, Shannon's asymptotic approach revolutionized digital communication and storage (despite the fact that all technological applications have very specific finite parameters). In physics, this approach is often called the *thermodynamic limit.* In mathematics, the asymptotic view is natural in discrete settings, like set systems and graphs (although some parameters of continuous structures are viewed asymptotically, e.g., dimension, genus, and continuity itself). Making discrete structures continuous and studying various limits uncovers hidden structure as well; perhaps more of that can be applied to the study of computation. The recent development of limit theories of combinatorial objects (see, e.g., the comprehensive book [Lov12]) is a promising example.

4. **Adversarial thinking**: *Prepare for the worst, replacing specific and structural restrictions and constraints by general, adversarial ones—more stringent demands often make things simpler to understand!* A recurring theme across ToC is that surprising algorithms and protocols are discovered when the models at hand allow stronger (rather than weaker) adversaries.[6] While seemingly counterintuitive, such a handicap (or worst-case view) often shines a light on an idea that may be obscured by specific details. And clearly, positive results (namely, upper bounds and algorithms), if they exist in such generality, are very desirable. Of course, equally often, lower bounds are found in general adversarial settings. Such negative results call for formulating more specific assumptions, under which the problem we care about does have a solution. But even then, guidance in modeling on how to restrict adversaries and avoid hardness can often be gleaned from understanding how and why algorithms fail in the presence of more general adversaries. The theory of cryptography in particular, where adversaries are typically restricted solely in their computational power, has been extremely successful due to this approach, which perfectly fits computational complexity theory.

5. **Classification**: *Organize computational tasks into (complexity) classes according to the amounts of various resources they require in various models.* Classification is natural across the sciences, but it is most often based on structural properties of objects. What is seemingly surprising in computational complexity is how a huge number and variety of problems snugly fit into relatively few complexity classes characterized by resource requirements, as well as the strong connections between classes defined by different resources. Of course, central open questions of the field are mostly about proving that certain pairs of classes are actually distinct. One could imagine the potential power of this type of computational complexity–based classification in other sciences (e.g., of synchronization or coordination processes in natural distributed systems, or of word problems and isomorphism problems on groups and algebras and manifolds in mathematics).

---

[6]These adversaries can be inputs, distributions on inputs, schedulers, noise, eavesdroppers, and so forth, depending on the context.

6. **Reductions**: *Ignore your ignorance, and even if you can't efficiently solve a problem, assume that you can, and explore which other problems it would help solve efficiently.* Despite furious arguments between philosophers on the power of "reductionism" to understand the world, understanding complex phenomena by breaking it into simpler parts is often responsible for great scientific discoveries. In computer science, a standard programming practice calls for using subroutines for a certain solved problem A as part of an algorithm for another more complex problem B. Algorithmic techniques make important use of such reductions. The main twist in ToC is the use of reductions for proving *hardness* as well as easiness: in the above example, not only does the easiness of A imply the easiness of B, conversely, the hardness of B implies the hardness of A. These relations produce partial orders of computational tasks under various notions of difficulty, which often also relate different models to one another. The sophistication and intricacy of reductions (which are themselves algorithms) have been raised to an art form in areas like cryptography and pseudo-randomness, where many other properties besides efficiency restrict the processes A and B. Different, mathematically rich sophistication arises in hardness of approximation reductions and PCP constructions. I have no doubt that far more structure and connections can be uncovered in fields like mathematics and biology when the language of reductions is systematically used to relate disparate problems and models.

7. **Completeness**: *Identify the most difficult problems in a complexity class.*[7] Combining items 5 and 6 above, the following phenomenon has been amazing initially, and has become a natural expectation by now: *whole complexity classes of problems that can be solved in certain limited resources and environments, can be "captured" by a single "complete" problem in the class.* Such *completeness* promises (via reductions) that better algorithms for that single problem will immediately entail the same improvements for all others in the class. And conversely, to separate one class from another, it suffices to prove hardness for that single problem. For both directions, one is free to focus on any complete problem when studying the whole class. Some notions of completeness, especially $\mathcal{NP}$-completeness, turn out to be very widespread phenomena across many disciplines. Finding more examples of such phenomena in mathematics and science for other notions would be extremely interesting and useful.

8. **Hardness**: *Prove intractability results—these are useful!* The potential utility of tractability results (namely, efficient algorithms) is obvious. However, knowing that a task is difficult (has no efficient algorithms, for some model and resource) can be equally useful! It can suggest changing the model and the definition of the task in numerous practical and scientific applications. And as we have seen, hard problems can be directly used to yield positive applications, as in cryptography and pseudo-randomness. Finally, failed attempts at proving hardness have suggested surprising algorithms that may not have been discovered otherwise (a famous example is Barrington's algorithm [Bar86]). Hardness of course is typically hard to prove, and conditional hardness is the next best thing; in this case, one naturally strives to minimize and simplify the assumptions needed.

9. **Barriers**: *When stuck for a long time on a major question, abstract all known techniques used for it so far, and try to formally argue that they will not suffice for its resolution.* Introspection has been a strong suit in computational complexity, and the formal study of barriers to progress on the major questions of the field has become part of the field itself. It is interesting that these investigations have often led to *computational* characterizations of proof techniques (even though these are what we researchers develop, not the computational models we study). This results in some unexpected connections between proofs and computations, and surprising unconditional results, for example, that no *natural*[8] proof exists of the hardness of factoring integers. Barriers are good not only as explanations (or excuses) for failure in resolving major open problems—they will also hopefully direct us to develop new, different techniques that bypass them. This has happened in the past, both in barriers to lower bounds (such as diagonalization) and barriers to upper bounds (like integrality gaps for linear or semi-definite programming). It would be interesting to see mathematical analogs of barriers for proving, for

---

[7]Namely, those which all other problems in the class reduce to in the sense above.
[8]In the formal sense of Razborov and Rudich [RR97].

example, the Riemann hypothesis using current techniques, which are similar in spirit to the barriers we have in computational complexity to proving $\mathcal{P} \neq \mathcal{NP}$ using current techniques.

10. **Play:** *Forget reality, ask for the impossible*. Despite being grounded and strongly connected to the practice (and demands) of computer technology, some of the greatest advances and innovations of ToC came from completely ignoring realistic constraints and intuitive biases about what is possible, and making up toy models and problems to play with and explore. It paid handsomely, both intellectually and practically, to seriously and thoroughly explore ideas and notions that seemed unreasonable or even outrageous when introduced. Examples include nondeterministic machines, proofs that are not always sound, placing inputs on players' foreheads, playing poker over the telephone, playing Chess and Go on an $n \times n$ board, pricing anarchy, counting without counters, conviction without knowledge, quantum post-selection, perfect randomness, anonymous ownership, putting sudoku and theorem proving on equal footing, and many many others that need more than a few words to describe. These efforts lead to continuously making up new games and rules and tasks for Alice and Bob, Arthur and Merlin, Byzantine generals and dining philosophers, multi-arm bandits and numerous players with less catchy names who are playing and analyzing these games. Of course, many of these games were inspired by external, realistic considerations, and the strong intuition of their inventors, but their abstract and free exploration was often essential for understanding the original applications and often completely unexpected ones. The fearlessness of making assumptions and exploring their consequences, or asking the impossible and investigating the minimal assumptions that will make it come true, has been a constant driving force of the field and will undoubtedly continue.

20.4   The computational complexity lens on the sciences

I now enter a topic that was hardly discussed in the book at all, with two important exceptions: $\mathcal{NP}$-completeness and quantum computing. On the first, I have already discussed at length in Section 3.10 how and why the notion of $\mathcal{NP}$-completeness has invaded all sciences and has been essential and ubiquitous in all. Let me summarize it here. "Ubiquity" may be a huge understatement in describing this rare scientific phenomenon. Scientists of all fields find the need to understand and use a very *technical* notion of computer science, and they publish numerous[9] articles that associate NP-completeness with whatever they are studying. Now on to quantum computing (which Chapter 11 is devoted to), by now an exciting, well-developed area that has sparked remarkable scientific interactions as well as billions of dollars in the development of technologies for building a quantum computer. This field serves as a perfect demonstration of what happens when the ToC methodology described in Section 20.3 is applied full force to the initial suggestions of the physicists [Ben80, Fey82, Deu85] about building quantum mechanical computers.

We now proceed far beyond these important examples.

A famous article of Eugene Wigner [Wig60], whose title captures its essence, is *The unreasonable effectiveness of mathematics in the natural sciences*.[10] Well, the effectiveness of ToC in the sciences *is* extremely reasonable, expected, and natural, and we will give plenty of demonstrations below. This view is completely consistent with the mechanistic view of the world, underlying most scientific theories. It became known as the *computational lens on the sciences*, or sometimes, when quantitative evaluation of efficient resource use is primary, the *computational complexity lens on the sciences*.

We will discuss numerous connections and interactions of ToC with diverse scientific disciplines, which focus on the integration of algorithmic and computational complexity considerations as an essential part of *modeling* and understanding nature. (This focus goes far beyond the vast use of computation as a *tool* in science, which, as mentioned, is of major importance in itself.) I see these activities as promising radical changes in many existing theories and major progress in better understanding many natural phenomena.

---

[9]You can amuse yourselves with Google Scholar, to see that "numerous" may be an understatement as well. Searching when the phrase "$\mathcal{NP}$-completeness" occurs *concurrently* with *each* of "physics," "biology," and "chemistry" in scientific publications gets tens of thousands of hits. To make these numbers even more meaningful, these pairs *each* occur very roughly as often as natural pairs as "energy, physics," "atom, chemistry," "life, biology," and "function, mathematics."

[10]This article was debated and interpreted by many after being published.

Before starting, let me point out that the computational element in scientific theories and models was always present (though often only implicitly). This followed both philosophical and practical considerations that long predate computational theory. First, while philosophers and scientists debated for centuries the precise properties of a scientific theory, all agree it should be *predictive*, namely, be able to supply (or guess well) the result of a new experiment before it is carried out. Without such predictive power, a theory is typically useless. Moreover, this predictive ability is necessary for the requirement of *falsifiability*: the existence of potential experiments that prove the theory wrong. But prediction is patently a computational task! Initial data is fed to the model as input, and the expected result must be supplied as output (e.g., "at what time will the sun or the moon rise tomorrow over New York?"). If this computational task is impossible or even merely intractable, then again the theory is useless. The need to solve computational problems to make scientific predictions was an essential component of the efforts of great scientists (as one famous example, Newton's *Principia* contains ingenious efficient algorithms for such prediction tasks). What has changed with the arrival of ToC and computational complexity is the ability to mathematically formalize and assess the tractability of these computational tasks.

Again, the pioneers of the field, Turing and von Neumann, already had a clear vision of natural processes as computational ones and realized that this view is essential to understanding nature. And as was typical of these giants, they did not deal with trifles. In one of the most-cited papers in biology, "A chemical basis for morphogenesis," Turing [Tur52] set out to give a "model of an embryo," that would explain the emergence of *structured* asymmetry and inhomogeneity (dominant in every living organism, with catchy examples like the color patterns on zebras, or left- and right-handedness) in a process that starts from symmetric, homogeneous initial conditions and follows symmetric evolution rules. Von Neumann was even bolder. In his book *The computer and the brain* [VN58], he built the first significant bridge between computer science and neuroscience. In this remarkably prescient document, von Neumann compares and contrasts computers and brains, and he uses Turing's universality, the digital nature of neurons' outputs, and the most up-to-date knowledge in both (young at that time) fields to show how much they can gain from interaction. In his book *Theory of self-reproducing automata* [VNB$^+$66], he takes on the ultimate challenge: comparing machines and living creatures, and modeling life, evolution, and reproduction. His *universal constructor*, a 29-state (universal) cellular automaton capable of self-reproduction essentially uses Turing's duality of program and data for replication, predating the soon-to-be-discovered[11] double-strand structure of DNA, in which the same duality is used in natural reproduction. It seems almost sinful to give one-sentence summaries of these detailed and ambitious seminal works, and I hope the reader will find out more, and especially appreciate how natural computational modeling is for complex biological systems.

It took the theory of computation several decades before plunging back in earnest to continue in these giants' footsteps and integrate the computational approach into a much wider variety of scientific disciplines. These decades were essential for the internal development of ToC: Many new computational models and types were studied, and the focus of computational complexity created a far better understanding of efficient utilization of various resources by various algorithms, and of arguing the limitations and trade-offs of that efficiency. This fit perfectly with the modeling of nature, in which efficient utilization of resources is the law of the land. The past three decades have seen a boom in the invasion of algorithmic thinking and computational modeling into many fields in the natural and social sciences. In many cases, visionary leaders of these fields have recognized the importance of the computational lens and suggested integrating its study; famous examples include Herb Simon's *bounded rationality* (see [Sim57]) in economics and Richard Feynman's *quantum computing* (see [Fey82, Fey86]) in physics. In many other cases, either following such cues or independently of them, visionary leaders of algorithms and computational complexity, armed with the powerful methodology and knowledge of this field and with a passion for understanding another, have proposed the integration of the computational view and classical scientific questions.

What is exciting is that some of these have turned into real interactions, with growing collaborations, joint works, conferences, and experiments. Of course, the difference in cultures and language, knowledge barriers, and simple caution and conservatism make some of these interactions move slower than others. Also, each of these fields is typically vast compared to ToC; this often limits interaction to specific subfields

---

[11]This book was published long after von Neumann's death.

and problem areas. Still, there are already plenty of stories to tell.

There is no doubt in my mind that the algorithmic lens on sciences is the global scientific paradigm shift of the twenty-first century. We are only witnessing its beginning. The computational models proposed will evolve, and new ones will be born, with more interaction, with novel experiments they will suggest, and with the absorption of computational methodology. These will naturally allow experimental sciences to become more theoretical, create further connections with mathematics, and interact even better with the complementary revolution of automated scientific discovery. This development will necessitate the education of every future scientist in ToC.

The following stories give some illustration of computational and algorithmic modeling in a variety of different areas and problems across a remarkably diverse set of scientific disciplines. This selection is based on my biases and limited knowledge. I focus mostly on outreach of ToC researchers to these fields. These stories should suffice to give the reader a sense of the scope of this interaction and its potential. There is no reason to think that any of the models or theories proposed are "correct" or final. Instead, the point is that computational complexity is an essential ingredient in them, and that this is typically a novelty that adds a new perspective to important problems in each field. As these collaborations expand, I view the stories here as a few drops in a huge bucket to fill. And yet, the combined impact of a very few decades of such interactions is, in my view, amazing!

Note that I will *not* discuss at all the many important ways in which the direct use of algorithms, computer systems, and technology interact and impact the sciences, including analyzing and interpreting vast amounts of scientific, social, and medical data and simulating scientific models and processes. This interaction is yet another revolution, which is beyond the scope of this chapter.

20.4.1   Molecular biology

Perhaps the fastest, largest, and most important growth of interaction between biologists and computer scientists has started with the human genome project at the turn of the millennium. The combination of new sequencing technologies with new efficient algorithms to analyze the outputs these produced gave birth to *computational biology* (or *bioinformatics*) (see, e.g., [Pev00, JP04] for earlier works on this collaboration). These fields grew to encompass all aspects of molecular biology and genetics, disease discovery and drug design, with similar collaborations combining modeling and algorithms, both human made and of the machine-learning variety. These collaborations, in academia and industry, probably dwarf in size everything else I mention combined!

In a reverse twist, computer scientists [Adl94, Lip95] initiated and studied *DNA computing*, the potential use of large molecules like DNA to more quickly solve difficult computational problems. Despite the slower speed of molecular processes in comparison with electronic ones, it seemed to offer immense (albeit, constant) parallelism at very low cost. This was indeed demonstrated in the lab by Len Adleman in his pioneering paper [Adl94]. Note that the universality of this computational model was not obvious, but was demonstrated in vivo by [BGBD$^+$04], with potential medical applications to cellular-level diagnosis and treatment.

Besides computing functions in the usual sense, Ned Seeman [See04] envisioned the design and physical construction of nano-materials from initial DNA and other molecules. Figure 23 demonstrates our ability to program DNA-like molecules to have certain atoms in certain positions that will interlock Lego-style with others to yield many different structures. These structures are completely unlike anything nature does on its own (e.g., a cube made by the carefully designed DNA strands in Figure 23). See also [RPW04, BRW05] for more algorithmic techniques and resulting structures. The ability to program such nano-level organic designs for a variety of functional and structural medical purposes is extremely promising (see, e.g., [KSZ$^+$10]), especially as one can build not only rigid structures but also moving machines (see, e.g., [GV08]). The same may be said of nano self-assembly for other materials (e.g., carbon-based; see Chapter 11 of [TPC15]). And indeed, there has been a growing flurry of joint activity of biologists and computer scientists on models and algorithms for "programmable matter" (see, e.g., the workshop report [DK16] and the thesis [Der17] for recent examples). In all technologies, tolerance of errors is a key issue, and fault-tolerant self-assembly is a major thrust in this effort—see the survey [Win06].

It is clear that these exciting abilities beg a general theory of the underlying "programming languages,"
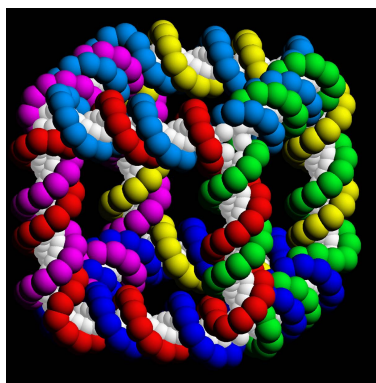
Figure 23. Cube from DNA strands. Used by permission of Nadrian C. Seeman.

whose basic building blocks are nano-materials, and whose basic operations use chemical or physical bonds. Studying the expressive power (namely, which architectures are possible and at what cost) of such languages (and ultimately, designs) is certainly on its way as exemplified above, in parallel with the technological developments. These models create new challenges for ToC researchers. An analogy which comes to mind is *origami*, the art of paper folding. For centuries ingenious artists found ingenious designs of ever more complex figures by simply folding a piece of paper with differently colored sides. And then came a bunch of computational geometers, who proved in [DDMO04] that *every* legal design is foldable, and moreover gave an efficient algorithm to do so! The wonderful book of Demaine and O'Rourke [DO08] describes such complete understanding of "physical" algorithms, some very relevant to the biological and chemical problems above.

20.4.2   Ecology and evolution

The fact that computational complexity is essential for models of nature was already clear to Charles Darwin in his *Origins*, and this was at the heart of one of the most interesting controversies in the history of science! Darwin took pains to approximate the age of Earth, in order to check whether his theory of evolution is consistent with the time it took life on Earth to reach such a level of diversity. In his day the (religious) dogma was still that the Earth is 6,000 years old, and it was clear to Darwin this was far from sufficient for mutations, reproduction, and natural selection to reach that diversity from one common origin. Luckily, the science of geology was being developed just then, and Darwin used it to estimate Earth's age at a few hundred million years, which seemed satisfactory to him.[12] But he became greatly upset (as he confessed in a letter to Wallace) when learning that the great physicist William Thomson (later Lord Kelvin) had estimated the age of the sun (based on its energy, according to the best physical theories of the period) only at a few tens of million years (roughly a factor of 10 *lower* than Darwin). This time estimate seemed to Darwin as having the potential to falsify his theory. Of course, as we know, Thomson was very wrong (as nuclear forces were discovered much later), and the age of the Earth is roughy 10 times *higher* than Darwin's estimate. But the message of this story is completely clear. Any natural process expends resources (like *time* in the story above), and so "computational complexity" must be an integral part of the relevant theory's consistency (and falsifiability). Indeed, better computational modeling of evolution processes and their complexity may lead to a revision and better understanding of this celebrated theory.

Building a quantitative, computational theory that will explain the complex mechanisms that evolved (e.g., in the cell) from simple ones through variation and selection has been the quest of Les Valiant in the past decade, starting with his paper, "Evolvability" [Val09]. He views evolution as a restricted form of his PAC *learning* methodology, which we examined in Chapter 17. Rather than going into any details, let me suggest Valiant's exciting and extremely readable book *Probably, Approximately Correct* [Val13], in

---

[12]Needless to say, this time estimate and its sufficiency for Darwin's theory relies on many assumptions, which he gave in detail.

which he expounds his theory, and in particular the notion of *ecorithms*: algorithms that interact with their environment.

From this very general question and model for evolution, we turn to discuss two specific conundrums, whose resolution I believe will involve computational complexity. Some initial suggestions in this direction have already been made.

The celebrated "problem of sex" (called the "queen of problems" in evolutionary biology by Graham Bell) asks to explain the prevalence of sexual reproduction in nature, in the face of its cost in energy and the loss of genetic material (which may have been improving by selection) in the process. After all, asexual reproduction seems far cheaper energetically and can continuously improve fitness. Of course, these simplistic considerations were only the beginning of a lengthy historical debate and many theories that are far from resolved. Here I would only like to point out, again without going into detail, recent contributions toward a quantitative, computational, and very different proposal by a combined team of evolutionary biologists and complexity theorists [LPDF08], which was further explored and expanded in the following (and other) papers: [LPPF10, CLPV14, MP15, LP16].

Yet another mystery of evolutionary biology is the "problem of conflict." Roughly, it asks how is it that creatures, optimized by evolution over millions of years' experience, can "freeze" by an inability to resolve a conflict between very different alternatives. This phenomenon has been observed in many species and situations (humans are extremely familiar with it), and as with the problem above, received plenty of debate and the proposal of differing theories. Another collaborative work [LP06] suggests an intricate computational model in which evolution of a system toward optimal behavior may organically create within itself two subsystems that can be at odds with each other! Computational limitations play a crucial role in this model.

Concluding this section, let me discuss the very general problem of understanding some very different algorithms of nature from a very different angle.
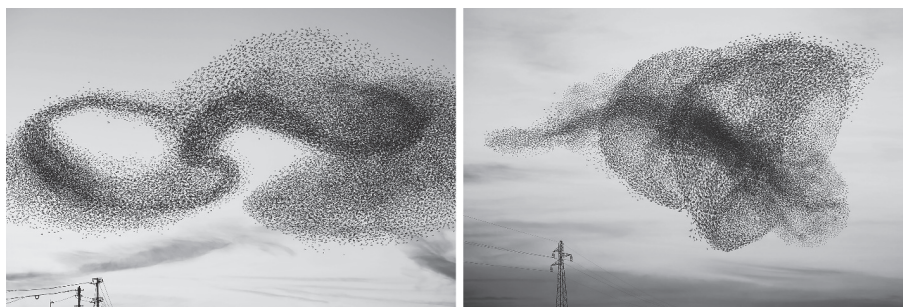
Figure 24. Starling murmuration (images by Alain Delorme).

The still pictures in Figure 24 do not even begin to make the impact of the amazing videos you can (and should) find by Internet search of "starling murmuration". It is nearly impossible to watch these videos of thousands of birds without, following disbelief and awe, seeking to understand the mechanism that allows this distributed system to produce such spectacular complex yet coordinated behavior. And indeed, for decades, these (and numerous other forms of remarkable coordination and action by many other species) have been studied by scientists and mathematicians. I point out one very general discrete model of such phenomena, called *influence systems*, that was proposed by Bernard Chazelle (see his surveys [Cha12,Cha15]). It attempts to capture the dynamics of a distributed network, in which the dynamic itself changes the network topology. This model is relevant not only to the natural systems and algorithms but also to many dynamic social situations. Chazelle's work develops techniques to study such nonlinear models and gives the first quantitative upper and lower bounds on convergence times in bird flocking and similar systems.

20.4.3   Neuroscience

Unlike many other scientific problems discussed in this chapter, the quest to understand the brain (especially the human brain!) always had a significant computational component, as the brain is generally regarded as an (albeit terribly complex) computational device. Very early works in computational modeling of the brain include McCulloch and Pitts' work [MP43] on *Nerve nets* (a collaboration of a neuroscientist and a logician) and the aforementioned book by von Neumann on the subject [VN58]. Great advances in biological techniques and in computational power have led to better and better understanding of neurons, the connections among neurons, and the activities of neurons during a variety of behaviors. These in turn have led to many (neural) network models of the brain, and of parts of the brain, attempting to explain various cognitive and other functions at different levels of specificity. It is interesting that the nature of many such models and the challenges in their analysis drew the interest and collaboration of physicists with biologists, which has been going on for several decades. All this represents vast and still rapidly growing work. Among numerous texts on the subject, an excellent comprehensive book surveying descriptive, mechanistic, and interpretive computational models of the brain at all levels is Dayan and Abbott's *Theoretical neuroscience* [DA01].

In what follows, I describe some recent interactions that stress the integration of computational complexity into models of the brain. I expect these interactions to grow and to lead to more collaboration and understanding.[13]

In terms of a general computational model of the brain, Les Valiant [Val00] pioneered a comprehensive design from a computational complexity perspective. It postulates precise mathematical assumptions about the interconnection patterns among neurons and the computational capabilities of individual neurons, which are consistent with known physiological results. It then proceeds to describe detailed algorithms that could be implemented in this model and to carry out some basic cognitive functions of memorization and learning. These details include how and what information is stored in (collections of) neurons, and a careful accounting of the physical and computational resources expended by these algorithms. To me, more than an actual model of the brain (which will probably take decades to figure out), Valiant's is a model for modeling the brain. In particular, it suggests further experiments and is eminently falsifiable. Curiously, it even predicted the need of certain physiological phenomena (e.g., strong synapses, neurogenesis) that some of the algorithms required, and these predictions were indeed verified to some extent after the first publication of the book. Valiant further articulates principles for general modeling of the brain in [Val06].

Other works attempt to computationally address more specific areas of the brain and focus on the implementation of more specific cognitive functions. For example, the hippocampus and memorization are further studied in [Val12, MWW16], the cortex and learning in [Val14, PV15], and the compression of information in neural tissue in [AZGMS14]. While these papers are mostly by computer scientists, they appear in journals read by (and presumably reviewed by) neuroscientists, and promise future collaboration.

One of the greatest computational and algorithmic challenges in neurobiology arises from *connectomics*, the field which (in some analogy with the human genome project, only orders of magnitude harder) attempts to map out the entire human brain, complete with all synaptic connections. Even partial success will hopefully give the ability to simulate its activities on a computer and to better understand the brain's processes and diseases. The very cross-disciplinary *blue brain project* [Mar06] represents perhaps the most massive such collaborative attempt today. The "big data" challenges arising even in the decisions of which data to collect, which to keep, how to organize it, and how to go about extracting the required structure out of it are great and far from agreed on (of course, these challenges are present and increasingly so in many other areas in which unprecedented amounts of data can be easily collected). Joint work of neuroscientists and computer scientists [LPS14] attempts to articulate principles guiding this task and possibly to build a computational theory of connectomics.

20.4.4   Quantum physics

In contrast to Section 20.4.3, where I focused on contributions of complexity theorists to neuroscience, here I focus on contributions of physicists, who use complexity-theoretic results and methodology to suggest resolu-

---

[13]See, for example, `https://simons.berkeley.edu/programs/brain2018`.

tions of the conundrums of physics. *As we shall see, the physicists' quest for understanding the fundamental structure of physical space and time may need the fundamental understanding of the computational resources of space and time!*

I have devoted all of Chapter 11 of this book to quantum computation, which was started by physicists and picked up by computer scientists. The field has led to a remarkably intense and productive collaboration between the two groups and has produced new results and fundamental questions in quantum mechanics and quantum information theory. A by-product of this interaction is the integration of complexity theory concepts and results by physicists, which allows for the proposals I will now sketch. I will focus solely on quantum gravity, or the long-sought theory that will enable the marriage of quantum mechanics and general relativity, moving toward Einstein's dream of a *unified theory*. And I will only discuss a few (related) examples. I fully expect far more integration of computational considerations to theories in all areas of physics, and far more interaction among the fields.

It is all about black holes. If the following sounds too mysterious and far-fetched to you, know that I feel the same, and so do some physicists. We discuss physical theories that are far from being testable in the foreseeable future, and arguments regarding them rest heavily on thought experiments. Moreover, many of these ideas are recent and are extensively debated. I will be lax and informal. For the brave who wish for detail, a technical but very readable survey, with plenty of intuition, detailing most aspects of the discussion below was written by Harlow [Har16]. It also contains all relevant references. A more informal survey that addresses these issues (with more computational complexity background) is the lecture notes of Aaronson [Aar16a]. Now brace yourselves for two stories full of buzzwords, which you should accept and read on as you might with fantasies, to the (computational) punchline at the end.

A fundamental connection of gravity and quantum mechanics is *Hawking radiation* (while Hawking was the first to fully develop it mathematically, he used the earlier ideas of Zeldovich and of Bekenstein). These physicists discovered that, although black holes are so heavy as to swallow everything around them, including light (which is the reason we don't see them directly), and seem doomed to grow forever, quantum effects nevertheless cause them to leak particles, and despite the extremely slow pace of this effect, black holes will eventually evaporate altogether!

Hawking radiation, which is a well accepted phenomenon, makes it extremely difficult to stitch the seemingly conflicting constraints imposed on it by general relativity (e.g., the singularity at the center of the black hole, the smooth structure of space-time at the event horizon[14] of the black hole) on the one hand, and those imposed by quantum mechanics (e.g., unitary evolution of the radiation, complementarity, no-cloning of quantum states, and the monogamy of entanglement) on the other. These potential conflicts between constraints (not all of which share the same acceptance status) are explored through ingenious thought experiments. The central one at hand, called *the firewall paradox*, was suggested by Almheiri et al. in [AMPS13]. Oversimplifying, the paradox can be roughly summarized by the ability of an observer sitting outside the event horizon to extract a qubit of information from the leaking Hawking radiation, then jump into the black hole and extract that *same* qubit after crossing the horizon,[15] which is a contradiction (it would violate a basic theorem of quantum information theory called the *monogamy of entanglement* mentioned above)!

Many proposals to resolve the paradox were suggested, but the one of Harlow and Hayden [HH13] (see elaboration and simplifications in [Aar16a], lecture 6) is unique in introducing a computational viewpoint, which is relevant for many other issues regarding such processes. It essentially views both the black hole producing the radiation from the information that fell into it, as well as the observer who extracts the paradoxical bit from that radiation, as efficient (polynomial-time) quantum algorithms. This view leads to a nontrivial computational analysis whose upshot is that if *quantum statistical zero knowledge is hard for quantum algorithms*,[16] then *the black hole will evaporate long before the observer can compute any paradoxical bit*. In short, computational complexity offers one resolution of this paradox: Time is too short for the

---

[14]The "boundary" of the black hole; anything inside it is trapped and falls into the black hole.

[15]I did not make this up!

[16]No need to understand what that means, other than that it is a computational assumption about the intractability of a natural problem, like $\mathcal{P} \neq \mathcal{NP}$. But I stress that this strange computational assumption, merging cryptography and quantum computing, has made it to the consciousness of physicists.

experiment to take place, simply because the observer's task is computationally hard.[17] This explanation (that in itself is conditional) is still debated along with others, but its computational spirit has entered the discussion, and with it the understanding that viewing nature and observers as algorithms is illuminating!

Understanding the processes in the interior of a black hole, which are subject to general relativity, is of course problematic almost by definition—there is no way to peek inside. Would it not be wonderful to deduce it from what we observe on its boundary? A remarkable proposal of Maldacena [Mal99] suggests something of that kind: a general method for accessing information about the behavior of particles governed by a quantum gravitational theory (like string theory) inside the "bulk" (or interior) of a space-time region, from studying the dynamics of a quantum mechanical system on its boundary. This theory has become known as the *AdS/CFT correspondence*, where "AdS" stands for anti de-Sitter space,[18] and "CFT" stands for conformal field theory.[19] Conformal field theories are completely accepted, and computations of their evolving parameters are by now standard in some cases. Using these to infer various parameters inside the AdS universe (or indeed, going the other way, using general relativity in AdS to bypass complex CFT computations),[20] one needs a dictionary that relates AdS to CFT quantities. Indeed, Maldacena provides an extensive such dictionary. While there is no proof of this duality, it has become an extremely useful tool in many areas of theoretical physics, and it allows making predictions about the bulk behavior that may perhaps be tested.

Now, let us focus on *the Susskind puzzle*, which asks for the analog of an AdS quantity called *wormhole length*. This idea of "wormholes" goes back to Einstein and Rosen, and according to another exciting new theory of Maldacena and Susskind, wormholes are supposed to explain the nature of entanglement. Anyway, all we need to know is that in appropriate AdS universes, the "length" of wormholes grows linearly with time. Susskind's puzzle is to fill the dictionary with the analog quantity of the corresponding CFT. It is hard for any physicist to imagine *any* quantity in quantum evolution that scales linearly with time! But Susskind finds an answer by considering computational complexity. Discretize time and space in the given CFT, and you get a quantum circuit on some n qubits, which the evolution iterates at every step (say, with some fixed initial state ψ_0, e.g., all zeroes). This iteration produces a sequence of quantum states, ψ_t, after t steps. Susskind proposes that the quantum circuit complexity of ψ_t in the CFT[21] *is* the dictionary analog of the wormhole length after t steps in the AdS evolution. In other words, a fundamental, *physical* component of a theory of quantum gravity corresponds under this proposal to a *computational* parameter in CFT. Again, while still fresh and highly debated, the view of processes in these theories as quantum circuits, and studying their computational complexity, is entering the discourse of theoretical physics.

20.4.5 Economics

One of the great successes of ToC interactions with a "remote" discipline has been with economics, particularly with game theory. This field was born in the late 1940s with the celebrated magnum opus *Theory of games and economic behavior* of John von Neumann and Oscar Morgenstern, and with John Nash's foundational papers on equilibria. During the following decades, game theory has developed sophisticated and elaborate theories of markets and strategic behavior of rational agents in them. These theories manifestly aim to be predictive and to inform the decision making of strategic entities (from von Neumann's original motivation to play well the (frightening) cold war game with possible strategies including nuclear attacks, through the (potentially vicious) competition of different firms over market share, where strategies may be simply the prices of products, all the way to the personal and social interactions in numerous situations). In many theories and papers of the field, it was intuitively clear that real people or organizations, with

---

[17]In the same way as in the section about Lord Kelvin's calculations of the age of the universe could have killed Darwin's evolution theory, if they had been correct.

[18]To stress that the theory of gravity we employ happens on a negatively curved space, namely, has a negative cosmological constant. As an aside, notice that this certainly is *not* the case in our universe, whose expansion is accelerating, and so has a positive cosmological constant; this does not stop physicists pursuing first the full understanding of this "toy model."

[19]Namely, a quantum field theory equipped with a strong symmetry called "conformal invariance." Importantly, there is no gravity in CFT!

[20]In general, these require quantum computation.

[21]Namely, the size of the smallest quantum circuit generating that state from the zero state.

computational limitations, must actually implement the suggested strategies. Indeed, some algorithms were proposed, but relatively little was done to formally investigate their performance or more generally to integrate computational complexity into these models, partly because the relevant computational theory was missing.

This void started filling in the 1980s, when a theory of algorithms and complexity was quite developed, and interest in ToC to apply it to economics grew. This work has accelerated since the 1990s with the advent of the Internet, creating new markets in which the processing time of participants became a primary concern. The interaction with ToC and the integration of the computational and algorithmic elements into economic theories and models encompasses many aspects of the field. It is lucky for me that an excellent collection of surveys appears in the book *Algorithmic game theory* [NRTV07], and so I will briefly recount below some of the major themes (by now, full-fledged research areas) covered there; far more detail and missing references can be found in that book's chapters. Plenty more research and collaboration has happened in the decade since this book's publication, and I will conclude with one (very different) example of interaction between the fields.

- *Complexity of equilibria*. Equilibria in a variety of strategic situations (games, markets) embody rational solution concepts: They guarantee each individual is satisfaction with their payoff given the behavior of other parties, and so are sought as a notion of stability. Central theorems of game theory prove that equilibria exist in very general situations. But how to achieve them was rarely discussed. The pioneering work of Papadimitriou [Pap94] observes that the arguments used in such proofs are purely existential (e.g., topological fixed-point theorems), and that natural algorithms to compute equilibria require exponential time. He develops a complexity theory for economics and mathematical problems of this nature, namely, where solutions are guaranteed by various existential principles. He raises the possibility that computing Nash equilibria is a complete problem for the class PPAD (namely, as hard as finding a Brouwer fixed point). This was settled in the affirmative 15 years later [DGP09, CDT09], even for 2-player games, giving the first indication of the hardness of Nash equilibria, possibly limiting its value as an economic solution concept.[22] Proving this result required the introduction of the family of *graphical games* into game theory in [KLS01]. The "need" for efficient equilibrium concepts that may salvage this hardness has led to approximate versions of Nash equilibria that do have near-polynomial time algorithms [LMM03]. Moreover, hardness results and algorithms were then extended to a variety of other equilibria, most notably the Arrow-Debreu market equilibria. Approximation algorithms for such problems connected this field with optimization, leading to proofs that in many "real-life" situations (restricting utilities, preferences, etc.), equilibria *can* be computed or approximated efficiently.

- *Price of anarchy*. This is a completely different take on equilibria, one that could have been part of economic theory but somehow was not. Proposed in the seminal paper by Koutsoupias and Papadimitriou [KP99], the *price of anarchy* is the ratio between the cost to society of a "distributed" solution, given by the worst equilibrium that individual (selfish and noncooperating) agent strategies achieve, and the optimal "centralized" cost, attained by strategies assigned to them by a (benevolent and fully informative) entity. Numerous situations in which agents need to make decisions in utilizing shared resources (e.g., cars on roads, packets on the Internet) affect performance (and thus individual and societal cost), and they raise the question of how costly is this freedom of choice ("rational anarchy"). This paper has led to very surprising results showing that in many general situations, the price of anarchy is far less than what might be expected. One major general result of this type is [RT02], establishing that this ratio is at most 4/3 for the traffic flow on *any* network of any size, in which delay on a link is a linear function of its congestion. This bound remarkably matches the tight example known for decades in economics as Braess' paradox (showing that adding a link to a network can actually *increase* congestion), that is achieved on a 4-node network. Another example of a similarly surprising result, in its generality and in the bound on the price of anarchy it supplies, was

---

[22]Further hardness results based on various (so far, nonstandard) *cryptographic* assumptions have recently been pursued (see, e.g., [BPR15]). These results heavily rely on the reductions and completeness foundations of [Pap94] and their subsequent development.

discovered for the completely different setting of Walrasian equilibrium [BLNPL14]. We stress that while the questions of distributed versus centralized solution costs are nonalgorithmic in nature, they, as well as their solutions, arise naturally from ToC methodology (here mainly the use of reductions and of approximation guarantees, neither common in classical game theory). This book exemplified the power of ToC methodology in other non-algorithmic questions and answers that arise in mathematics. These important works have naturally led also to many algorithmic questions, again connecting with optimization, on-line algorithms, and distributed computation. These works connect with research questions in game theory and have generated new collaborations between the fields.

- *Mechanism design* (sometimes called "inverse game theory"). A major goal of game theory is understanding the behavior of rational, strategic agents participating in games which are already *pre-specified*. Mechanism design is the field of game theory which asks the meta-question: how to specify (or engineer) these games, incentives, and rules of behavior so as to force similarly rational, strategic agents to achieve certain global goals. Examples abound: governments setting up laws and distributing resources for various societal benefits, auctions aimed at selling items and maximizing profit, voting systems for best representations of public opinion, and healthcare and other social systems balancing services and costs. Many basic principles and theorems were discovered in this study, explaining, for example, how to elicit agents' actions to have various properties, such as truthfulness, maximizing profit, or some global social outcome. Again, the advent of the Internet has added numerous new applications for the design of mechanisms, which are also far more complex than many previously considered due to the potential huge number of participants, speed of action, and the distributed nature of interaction (the market of on-line advertisements is a perfect example). Now algorithms are the agents, and their advantages and limitations must be taken into account. This has greatly enriched the theory of mechanism design in manifold ways. For one example, the use of communication complexity and other tools allowed for hardness results limiting the power of mechanisms. For another, the view of (distributed) algorithms and protocols as executed by rational agents has added a completely new ingredient to algorithmic theory; its design should ensure behavior that makes it correct, efficient, fair, and so forth when implemented by selfish rational players. The collaboration between ToC and game theory in this field is perhaps the most extensive, and is manifest in industry as well, as one can observe even when restricting attention to the theory and practice of auctions for advertisements motivated by a multibillion dollar search-engine advertising industry.

- *Cryptography and game theory*. I mentioned above the negative news to economics following the computational complexity limitations on the discovery and implementation of desired strategies (e.g., in equilibria). However, plenty of positive news follows from cryptography on the implementation of strategies that seem impossible or hard to achieve in the (generally assumed) information-theoretic environment of game theory. As one example, take the notion of *correlated equilibrium* of Aumann, which in contrast to the Nash equilibrium is easy to compute, *if* all information about all players' utilities is known to some central trusted player. This "unrealistic" assumption renders Aumann's notion almost useless in situations where no such trusted authority exists or players wish to protect their privacy, until you realize that the *secure multi-party computation*, which I elaborated on in Chapter 18, solves just this very problem. Indeed, the papers [Yao86, GMW87, BOGW88] and their followers do so (in different settings), eliminating the need of a trusted party in any such situation, and making Auman's solution concept (and many others) eminently realistic and useful! The many differences of axioms of both fields (in crypto, privacy is a goal, whereas in game theory, it is often the means; in crypto, players may be honest or malicious, but in game theory, they are assumed to be rational, etc.) guarantee great synergy and collaboration as we integrate the point of view of one field into the other.

Let me conclude with an example of interaction between the fields that tackles a completely different, fundamental issue that probes the assumption of transparency and efficiency of markets. Mountains of articles were written about the financial crisis of 2008, proposing numerous reasons and models for that colossal collapse. There is a general agreement that the mis-pricing of financial derivatives, such as CDOs

(collateralized debt obligations) played a central role, and there are debates about how to fight it. One paper that proposes a computational complexity perspective on this situation is the collaborative [ABBG11], "Computational complexity and information asymmetry in financial products." In it, the authors propose a simple model that demonstrates how information asymmetry between buyer and seller, and hence the cost they assign to the same derivative, can be extremely high, even in a fully transparent market, and even if the buyer (or any inspector of the derivative contract) is extremely powerful computationally (e.g., a large bank, regulator, or the government). Creating such *artificial* examples is easy for anyone who has read Chapter 18, and indeed, as explained there, this very asymmetry of information underlies all Internet security and e-commerce systems! The paper [ABBG11] constructs simple, *natural* CDOs, made up by mixing "normal" and "junk" assets, and shows that distinguishing those in which the mixture is random from those in which the mixture biases for "junk" is a computationally difficult task under well-studied complexity assumptions. This paper has gotten some traction, and has been referenced in the economics literature (see, e.g., the very different [CKL13, BCHP17]).[23] I feel that the many financial models (e.g., those of markets and risk in the papers cited here) invite possible revisions and extensions in light of computational complexity theory. Unlike game theory, the finance side of economics has had precious few collaborations with ToC so far, but I expect these to grow.

20.4.6   Social science

The advent of the Internet and the many types of social activities and networks it supports has opened up both a host of new problems for social scientists as well as, for the first time, the ability to run social experiments on vast numbers of participants. This has created numerous new collaborations between computer and social scientists, which is enriching many aspects of social science and making it much more quantitative. Aside for the one (early) famous example below, which illustrates the nature of ToC influence, I will simply refer the reader to the excellent and extensive book *Networks, crowds, and markets: Reasoning about a highly connected world* [EK10]. This book itself is such a collaboration, and I note again that plenty more has happened in the 8 years since its publication. The variety of topics of interaction between ToC and social science described there is staggering (especially given the short amount of time since interaction started), and I give just a taste. These topics highlight a major evolution in ToC of the study of distributed networks. Most of this large field focuses on human-made design of networks and algorithms for them. The interaction with social science has expanded this field greatly! Among the many new aspects the book [EK10] describes are the growth and change of different (physical, information, social) networks, the organic appearance and evolution of power structures in them (e.g., hubs, authorities, institutions), and the nature of dynamic processes (gossip, influence, incentives, connectedness, and others) on networks. Many of the network models include game-theoretic and economic aspects, when viewed as markets whose participants are strategic agents placed at the nodes. This connects the research here with that of Section 20.4.5. In all, the ease of getting real-world data on many of these aids the testing of new models, theories, and algorithms. Of course, many other aspects and models of social interactions are not covered by the book [EK10], for example, the work of Chazelle on *influence systems* [Cha12, Cha15] mentioned above.

Our example is Jon Kleinberg's paper [Kle00], titled "The small world phenomenon: an algorithmic perspective." We only sketch it here; the reader is encouraged to read its lucid, detailed exposition and references. This paper is a take on Stanley Milgram's famous experiment on "the small world problem," first described in [Mil67]. Following previous work pursuing the popular cliche "six degrees of separation," Milgram designed an experiment to test the connectivity and distance between pairs of people in the United States by asking people to forward a letter from one to the other using only intermediaries who are on a "first name basis" with each other. The required chains were typically indeed very short, and a sequence of analytical network models were suggested and analyzed that might at the same time explain social relations in society and support such short distances. But it took more than 30 years before Kleinberg pointed out that there is another fundamental problem to be explained in Milgram's experiment: *Even if short paths exist, why were such paths found by a purely local algorithm*? Kleinberg went on to give a thorough answer.

---

[23]Be aware that the word "complexity" can mean many different things in economics.

First, he proved that many network models (including most suggested in past work) have an abundance of short paths, which *no* local algorithm will find! Next, he proposed extended natural models and identified the unique one in this class for which a local algorithm will succeed. This single work was key to the collaborations mentioned above, and to numerous follow-up works.

20.5   Conceptual contributions; or, algorithms and philosophy

Let me move up another level of abstraction when discussing the nature of ToC's impact. We have already seen some of the following important contributions in the book, but there are plenty more we have not discussed!

Alan Turing not only laid out the groundwork for the mathematical foundations of computer science and for the computer revolution that followed; he also demonstrated the powerful light that computation can shine on fundamental concepts. His article, "Computing machinery and intelligence" [Tur50], takes on one of the most difficult and controversial notions: *intelligence*. With remarkable clarity and brevity, rare in arguments on such philosophically, socially, and scientifically charged concepts,[24] he proposes a completely fresh and original approach to defining and understanding it.

Over the years, ToC was presented with the need to understand, and thus also to define, concepts and notions from many disciplines that have occupied intellectuals for centuries. This need arose sometimes from scientific or technological origins, and sometimes indeed from the philosophical inclinations of thinkers in the field. With the computational focus and language at heart, they gave birth to novel definitions of such concepts, breathing fresh ideas into arguments over their meaning and utility. This is far from saying that the computational lens is better than other views—it sometimes complements others. Instead, it points to the intellectual depth inherent in the field, and to the relevance of computation (and even more, of computational complexity) at the philosophical level. I feel proud to belong to a field that has seriously taken on defining (sometimes redefining, sometimes in several ways) and understanding such fundamental notions that include: *collusion, coordination, conflict, entropy, equilibrium, evolution, fairness, game, induction, intelligence, interaction, knowledge, language, learning, ontology, prediction, privacy, process, proof, secret, simultaneity, strategy, synchrony, randomness, and verification.*

It is worthwhile reading this list again, slowly. I find it quite remarkable to contrast the long history, volumes of text written, and intellectual breadth that the concepts in this list represent, with the small size and the relative youthfulness of ToC, which has added so much to their understanding. Moreover, for many of these notions there was little or no expectation that the science of computing would need to deal with them, and that if it did, would find meaningful and illuminating ways to do so. It is only with hindsight that it all looks so natural now. This book discusses some of these notions, and when it does, I have tried to articulate the way computational complexity illuminates them, often borrowing directly from the very articulate people who originated these definitions. Scott Aaronson [Aar13b] notes that although some of us convey extremely well the philosophical implications of our work to our own community, we perhaps should broadcast more of it to the outside world, in particular to philosophers. His paper tries to do just that—explain the potentially important role that computational complexity can play in understanding some philosophical and scientific conundrums.

This whole direction, abstracting the role of computation and complexity in the intellectual discourse, probably deserves another book, far less technical and more accessible than this one. I will content myself here by concluding with two very high-level philosophical issues, possibly metaphilosophical principles, that I did not discuss much in the book. The first is *subjectivity* and the second is *interaction*. Again, neither is new, but armed with the computational complexity lens, they inform and revise many of the notions mentioned above.

Subjectivity   The first is the role of computational complexity in establishing *subjectivity* in the view of reality. For centuries, properties (including many of the notions listed above) were defined and treated as intrinsic, *objective* properties of objects, agents, or interactions. Now they are treated by computational

---

[24]But typical of all of Turing's arguments on many other issues, including his definition of a computer!

complexity theory as *subjective*, highly dependent on the observer! This is reminiscent of Einstein's special relativity and its radical (at the time) predictions that different observers may see the same object as having different lengths or may disagree on whether two events are simultaneous. Einstein derived such predictions by adding a single axiom: that *the speed of light is constant*. The axiom taken by computational complexity is that *certain problems are intractable*. This has no less radical consequences. A random event can be completely unpredictable for one observer and be predictable for another. A message can be clear to one receiver, and completely incomprehensible to all others. These consequences of the simple computational axiom underlie Internet security and e-commerce, as well as many other applications. What is more, differences in the computational powers of different observers (who may actually be participants in a certain activity) allow us to *quantify* "how much" of a given property is present from their points of view. For example, we can quantify precisely, given the computational complexity of an observer, how predictable some event is, or how much knowledge the observer can glean about a secret message, or how accurately the observer can evaluate a complex economic commodity. In short, having different computational powers implies having different inference, analysis, and reasoning powers. The sophisticated use of these limitations, together with appropriate versions of our computational axiom, make this subjective view of reality the source of some of the most exciting and important applications of computing systems.

Interaction Interaction enhances many aspects in life. The importance of interaction for *understanding* easily follows by comparing a class in which students are allowed to ask questions to a class in which they are not. Interaction of this nature underlies Turing's "imitation game," which underpins his approach to *intelligence*. Simple examples also show that the *cost* (e.g., the amount of communication) of natural computational tasks between two parties can be much smaller in bidirectional versus unidirectional interactions. The study of interaction in computational complexity has uncovered extremely surprising powers, which are discussed in some chapters of this book. Here are some examples. One major impact was on the central notion of *proof*. Classically, proofs are unidirectional communications from a prover to a verifier. Allowing bidirectional *interactive proofs*[25] has led to a complete revision of the capabilities of proofs. First, more can be proved interactively: The $\mathcal{IP} = \mathcal{PSPACE}$ theorem [Sha92] basically shows how a prover can convince a verifier of having a winning strategy in, for example, chess, something unimaginable in classical proofs. Second, proofs can have paradoxical properties: The $\mathcal{NP}$ in zero-knowledge theorem [GMW91] basically shows that every statement that has a proof also has an interactive proof that is equally convincing but reveals no new information to the verifier. Other fundamental discoveries extend results from the unidirectional to the interactive setting. Shannon's famous result [Sha48] proves that unidirectional communication can be protected from a constant noise-rate with only constant redundancy (via his error-correcting codes). The same result holds (via Schulman's new ingenious codes [Sch92, Sch93]) for bidirectional communication, despite the adaptivity of such conversations.

Let me conclude this section with a a higher-level contribution, where the study of interaction (in the surprising context of information privacy) actually informs the *scientific method* itself. A key axiom of this method demands that a scientist decides the questions to ask about the data *before* she collects them. This forces a unidirectional communication with nature: first information is received or collected, and only then processed. Of course, findings lead to new questions and require new experiments and data acquisition. Scientists may be (indeed, some are) tempted, for efficiency reasons, to use existing data to answer the new questions, rather than collecting more data. Are the results of such adaptive use of data valid? As it happens, an interactive definition of *differential privacy*, and efficient algorithms for maintaining it, surprisingly inform the fragile validity of adaptive data analysis [DFH+15], showing that in certain cases, the above temptation may be justified, and how! Such deep connections, which go far beyond the intended utility of the models and algorithms developed, further show the depth of the very concept of computation.

---

[25]And incorporating randomness, error, and complexity.

20.6   Algorithms and technology

We now switch gears to discuss several aspects of the intense and continuous interaction between the theory of algorithm design and the industry of computing systems.

Given its focus on computational complexity, this book has paid little attention to *the* major effort of ToC, namely, algorithm design. A major product of ToC is the design and analysis of efficient algorithms for specific problems, and more fundamentally, the design of algorithmic techniques and paradigms for broad classes of computational tasks in a variety of models. Here I address a few issues related to algorithms, their design and impact, and their interactions with technology. One word of caution: In much of the ensuing discussion, the boundary between algorithms and technology is not always completely clear (as is the case between theory and practice, science and engineering, pure and applied math).

20.6.1   Algorithmic heroes

The combined power of algorithms and technology has provided myriad applications that has totally transformed human society in just a few decades. No doubt, this trend will continue and accelerate. And although both ingenious algorithms and remarkable technological advances played central roles in this revolution, it seems to me that the general public is not quite aware of the essential role algorithms play in enabling the applications they experience, and rather attribute the changes almost solely to technological advances. Of course, in many cases, algorithmic and technological advances are intertwined. But sadly this ignorance exists even when *a single algorithm has enabled a whole industry*, and even when the essence of that algorithm can be rather easily explained to any motivated listener with a basic math background.

In popular talks I give about algorithms and complexity, I show two lists of names, and ask the audience which of them is familiar. The first list has names like Johannes Gutenberg, Joseph Jacquard, Thomas Edison, and James Watt. These were presented as cultural heroes when I was growing up for inventions that dramatically advanced society at the time: respectively the printing press, the weaving loom, the light bulb, and the steam engine. Most general audiences recognize them and their inventions (although less so for young audiences). The second list I show has names like Edsger Dijkstra, Donald Knuth, John Tukey, and Elwyn Berlekamp. Here I mostly draw blanks. I then show, just for impression, the few lines of pseudocode for each of the respective algorithms—shortest paths, string matching, fast Fourier transform and Reed-Solomon decoding—that these algorithmic heroes and their collaborators developed and others extended. I note that like the older inventions of physical devices mentioned, these tiny intellectual creations are extremely brilliant and efficient, but that their impact may be far larger. To demonstrate this, I finally discuss (some of) the industries that have variants of these algorithms enabling them, respectively: automatic navigation, computational biology, medical imaging, and all storage and communication devices.

It seems to me that such stories are exciting and accessible school material even in middle school, and the theory community should have every incentive to bring it there. Of course, these are just examples, and there are many others. Prime examples of such algorithms include RSA, underlying most of e-commerce; PageRank, underlying Internet search; and back-propagation, without which the "deep-networks" revolution in machine learning (that we will soon discuss) would be impossible. In these examples, some people may recognize the names of the algorithms (or the buzz surrounding them), but not necessarily their inventors. Finally, there are even more basic algorithmic principles and structures for efficient data organization and access, including hashing, caching, sketching, heaps, trees, and bags-of-words, among many others,[26] all gems underlying many algorithms and applications.

20.6.2   Algorithms and Moore's law

Another point to discuss is the relative contribution of technology[27] and algorithms to applications, especially in the future. Clearly, a computational application becomes available only if it is efficient enough to be useful to its users and profitable to its developers and manufacturers. How much more efficient can we

---

[26]The reader can find details on all of these on Wikipedia and other Internet sources.

[27]Here, in the sense of physical infrastructure of computing systems.

make hardware and software? Focusing on the most basic resources, whose efficiency is paramount—speed and size—clarifies how incredible the science and technology of hardware design has been. In what became known as *Moore's law*, Gordon Moore predicted in the 1960s the doubling of the number of components on integrated circuits every 18 months. Similar predictions were made regarding the increase in computational speed. Miraculously (at least to me), this exponential growth persisted at the predicted rate for half a century! But not anymore.

Whether the recent slowing of Moore's law is a sign of its imminent death, or it will stay with us for a while longer, technology has *absolute* physical limits. Components will never be smaller than atoms, and communication between them will never exceed the speed of light. At that point, and likely much earlier, the *only* source of efficiency will be the invention of better algorithms: Which computational applications and products we will have available, and which we will not, will depend on algorithmic advances far more than on technological ones. Here, for numerous problems, the best time and space bounds we know seem far from optimal, and there seems to be plenty of room for improvements.

20.6.3   Algorithmic gems vs. deep nets

So, it is pretty certain that algorithms will rule Earth, but which algorithms? The remarkable computing power that technology has already provided us (together with very important algorithmic advances) has enabled a new breed of algorithms that arise from machine learning. In short, we now have algorithms as technological products. Tech and medical companies (as well as governments) invest enormous funds and effort in these new algorithms, which I now describe (and are discussed more generally in Section 20.7.2).

In stark contrast to the elegant, concise algorithmic gems mentioned above, which were devised by humans, many new algorithms simply "create themselves," with relatively little intervention from humans, mainly through interaction with massive data. This contrast with classic algorithm design is heightened as these new algorithms are typically (and perhaps necessarily) gigantic and very poorly understood by humans. I am referring of course to the "deep networks" (see [GBC16] for an extensive text on their uses in learning and optimization), a common name for heuristics modeled after networks of neurons.[28] I conclude this section with a few remarks on these algorithms and the challenges and questions they raise. Note that although machine learning was the topic of Chapter 17, that chapter focuses on theoretical models and simple tasks, and not on the complex tasks and the heuristics mentioned here.

These self-taught algorithms attempt to solve numerous problems that are often hard to formally define, such as finding "significant signals" in huge data sets that are often extremely noisy—be they financial, astrophysical, biological, or the Internet. The structures one may want to extract/uncover may be clusters, correlations, geometric or numerical patterns, or completely unexpected ones. These may represent, for example, familiar faces or objects in pictures, groups of friends and interests in social media, market performance of companies in the buying and selling of stocks, effects of treatments or genetic defects in biological data, and illuminating interactions of matter and energy in physical observations.

This is no place to give a proper description of deep nets and their training process. It suffices to say that today, their size can get to millions of gates and wires between gates. Each connection has a strength parameter that the training process attempts to optimize. In short, training is a huge optimization problem with an ill-specified goal and numerous degrees of freedom in the heuristics driving the attempted optimization. This "black magic" has worked extremely well only in relatively few cases so far. But in a growing number of cases, it greatly outperforms any human-designed algorithm. It is extremely interesting when such a self-taught program can label by name the essential content of arbitrary pictures taken by humans nearly as well as humans would. It is very impressive that another such program, after playing against itself a billion games of Go or Chess (knowing only the rules of the game) can now beat the world's best human players. Great progress by such programs is made on human language understanding and translation, and perhaps their fastest growth is felt in "data science," where these programs play ever more important roles in actual scientific discovery. Indeed, one wonders at what point they will be able to better

---

[28]There are many other types of heuristics, past or future, to which this discussion is relevant as well. However, deep nets seem to have become dominant in the past decade, so I'll focus on them.

articulate the new scientific laws uncovered and pursue their consequences, like medical drugs and treatments or food and energy supply, also without human help. There is no doubt that every person, every government, and human society as a whole, should give deep thought, consideration and discussion to the many societal questions, potential dangers and promises, raised by these remarkable abilities of machines (which are only beginning to unravel).

The main point I find fascinating and challenging on this front, and for which ToC can and should contribute, is a theoretical understanding of such algorithms. This understanding should include ways to make their training more principled and efficient, and developing models and tools to assess their performance and output quality. A major challenge is to understand why and for what tasks deep networks are successful, and what are their limitations. Further, it is of crucial importance, given their growing prevalence in systems humans crucially depend on, to explore how to make them *fair* (and what this means), how susceptible they are to adversarial data, and how to protect against such data. Of course, the size and complexity of deep nets may put limits to how well they can be theoretically understood (much like massive creations of nature, such as the brain and other biological systems). Indeed, it is not unlikely that the theoretical study of deep nets (on which, unlike animals, experimentation is free from the Declaration of Helsinki guidelines) will help in better understanding biological systems and vice versa.

Given our current ignorance (namely, gaps between upper and lower bounds) regarding many well-posed important problems, I have no doubt that there is plenty more room for the discovery of algorithmic gems that we can easily understand, appreciate, and use. To drive this point home, let me note in conclusion that no deep nets would exist without a few great algorithmic gems embedded in practically all of them, including the extremely efficient *back propagation* and *gradient descent* algorithms.

20.7   Some important challenges of ToC

Even at a high level, there are just too many important challenges of ToC to list here, especially in the wide contexts of its connections to the practical world of computing and its broad connections to the sciences. (Also, quite a number of important conjectures appear in different chapters of this book.) What I would like to single out here is four meta-challenges in the complexity-theoretic core of ToC, which have many incarnations and formulations (some discussed in the book). These (naturally interrelated) challenges are certifying intractability, understanding heuristics, strengthening the foundations of cryptography, and exploring the Church-Turing thesis.

20.7.1   Certifying intractability

By far the greatest challenge of computational theory (and a major challenge of mathematics) is to establish that *some*, indeed any, natural computational task is *nontrivially* difficult for a general computational model. This question has many incarnations, but the sad state of affairs is that even though almost every natural task we really care to perform seems exponentially difficult, we cannot establish even super-linear lower bounds (namely, that a given task is harder than just reading the input)! The $\mathcal{P} \neq \mathcal{NP}$ conjecture is of course the most popular manifestation of this challenge (e.g., proving a lower bound for some $\mathcal{NP}$-complete problem like the satisfiability of Boolean formulas or 3-coloring of maps). Similarly, establishing a nontrivial lower bound (if true) for even more basic (and practically burning) questions like integer multiplication or the discrete Fourier transform elude us as well. That for 80 years we have had a formal mathematical model in which to prove such lower bounds (and certainly great motivation for doing so) demonstrates the difficulty of this major problem!

Essentially the same sad state of affairs exists in questions of *proof complexity*: We know no nontrivial lower bounds on the length of, for example, Frege proofs of *any* propositional tautology. This quest is manifested by the $\mathcal{NP} \neq \text{co}\mathcal{NP}$ conjecture. Likewise, in *arithmetic complexity*, we know no nontrivial lower bounds on the number of arithmetic operations (sums and products) needed to compute natural polynomials like the permanent; this quest is manifested by the $\mathcal{VP} \neq \mathcal{VNP}$ conjecture. And the same holds for many other models and modes of computation. We can't prove hardness!

As discussed in Chapter 5, we have "excuses" in the form of *barrier results*, which give partial explanations of why current techniques fail to deliver nontrivial Boolean lower bounds. But we do not really understand why we have not yet found techniques to bypass these barriers. Furthermore, for proof complexity and arithmetic complexity, we don't even have any excuses in the form of barrier results. In all models, the lower bounds can be expressed quite neatly in various forms as combinatorial and algebraic questions that do not betray any clear distinction from many other such questions that were resolved in these or other fields of mathematics (and in particular, the same lower bound questions for restricted models). So, understanding *why* proving nontrivial lower bounds is so difficult is one of the greatest challenges of the field, and probing this mystery abstractly may lead to progress on proving them. Of course, two explanations must be mentioned, even though few (if any) believe them. First, that these conjectures are false, no lower bound exists, and everything is extremely easy to compute and prove. Second, that these conjectures are actually independent of the logical foundations of mathematics (say, of the ZFC axioms for set theory). So they can be true or false "as we please" from a provability standpoint, even though they have only one particular truth value in the "real world."[29] Ruling out the second option of logical independence, without resolving these conjectures, is an extremely interesting direction as well.

The simplest explanation (which is probably the correct one) is that proving lower bounds for general models is truly one of the deepest and most difficult mathematical problems we know. Thus, probably far more groundwork, on a variety of limited computational models, is needed to gain a true understanding of general computational models and their limitations. Of all challenges of the field, this is the one I care about most, and hope that it will continue to attract excellent researchers to develop the necessary structural theory and techniques for proving such lower bounds. I am certain that such understanding will not only expose the limitations of computation but also its full power.

20.7.2   Understanding heuristics

We often get much more than we (theoretically) deserve. Many algorithms are known to behave extremely badly in *some* instances, but have excellent performance on "typical" ones. Let's discuss some families of examples.

- Some algorithms for diverse problems seem to perform far better (in time spent or output quality) than their theoretical analysis (if any) guarantees. Prototypical examples commonly given of such *heuristics* include *SAT* and TSP solvers on huge instances, and the simplex algorithm for linear programming. Of course, these heuristics are very clever, carefully optimized, and fine-tuned.

- As discussed in more detail in Section 20.6.3, an explosion of heuristics arises from the new ability to train deep networks to solve learning, optimization, and game-playing problems. There is also in these cases a process of fine tuning, but it is largely automated in that mysterious, self-teaching "training" process. In many (but certainly not all) problems motivated by a variety of applications, these heuristics outperform carefully designed algorithms, or human experts, on many natural data sets.

- Another family of examples, of a very different nature, is the following. Probabilistic algorithms are designed to work correctly (with high probability), assuming that the random coin tosses they use are perfect (namely, independent and unbiased). But they often *seem to* perform equally well when fed a sequence of bits generated in some arbitrary deterministic fashion, or taken from physical sources like Internet traffic or user keystrokes. This (possibly surprising) phenomenon happens, for example, in numerous Monte-Carlo simulations used by physicists.

- Finally, nature itself provides us with many processes that *seem to* quickly solve "hard problems," like protein folding, formation of foams, market equilibria, and coordinated action of swarms. Here of course we typically don't really know nature's algorithms, but may have models for them, which in many cases fail to predict or guarantee the observed (high-quality) performance.

---

[29]This can happen, for instance, if a polynomial-time algorithm for *SAT* exists whose correctness is independent of ZFC.

The general but vague explanation for such phenomena is that the *inputs* to all these algorithms and processes come from certain sets or distributions (which nature or humanity generate), for which the given heuristic is far more efficient than for arbitrary inputs. The suitability of a heuristic to the "real-world" inputs it processes can have many reasons. In some cases, algorithm designers have managed to put their fingers on the actual (or related) input distribution and design the algorithm to be super efficient on such inputs (even though in general it may be slow), as happens for some TSP and *SAT* solvers. In some cases, algorithms were designed to be natural and elegant, like the simplex algorithm, where its superb performance on real-life linear programs was surprising and indeed lucky; our understanding of it is lacking, despite various later attempts to explain it. In yet other cases, algorithms *evolved* to suit their input distribution, as happens for many neural nets and other (supervised and unsupervised) learning frameworks. Moreover, in nature, where such learning and evolution abound, it is quite possible that in many settings, both the algorithm and the inputs *co-evolved* to yield superior performance (e.g., perhaps proteins that survived are in particular those that fold easily by the algorithm in our cells).

Understanding actual input distributions arising "in practice" for various computational tasks, and designing algorithms that handle them well (or explaining why certain heuristics do), is a very old and fundamental conundrum of the field. The many decades of algorithms research have provided theoretical models and frameworks that address this important issue from various viewpoints. General approaches include probabilistic analysis of algorithms [Kar76], semi-random models [FK01], smoothed analysis [ST04b], stable instances [BL12, BBG13], input distributions with certain moment bounds [HS17, KS17], and many others.[30] Some of these approaches were developed, extended, and applied significantly. Beyond such general approaches, which apply to wide families of algorithmic tasks, numerous papers address specific tasks, targeting and taking advantage of their "natural" input distributions. This interaction between modeling inputs and algorithmic design is extremely important and will hopefully lead to a better understanding of performance (and other behavioral properties) of such algorithms on "relevant" inputs.

With the advent of machine learning algorithms that interact and improve with the data, there are more and more heuristics whose performance (which in many applications is extremely impressive) is hard to explain. These and other existing heuristic mysteries present a great challenge to developing evaluation criteria for the actual performance and behavior of such algorithms, which we have already discussed in Section 20.6.3. Possibly missing is a theory of benchmarking, which will allow comparing heuristics and suggesting rational guidelines for choosing the many parameters of learning systems before setting them loose on the data. Certainly missing is a theory explaining the kinds of problems and kinds of data that, for example, deep-nets are good at solving, those for which they are not, and why. The need for such a theory is not only theoretical! We will soon have to *trust* such algorithms (when they'll replace humans, e.g., in driving cars and providing medical diagnoses and treatment), and trusting algorithms we do not understand is a very different matter than trusting algorithms we do understand. Most computer systems in existence are of the latter type, but the balance is shifting. I believe that (beyond performance) theories modeling "social aspects" of program output should and will be developed, and some guarantees will have to be supplied by heuristics employed in the public sphere. These issues are already entering public policy and legal discourse (for one fascinating example, on the idea of delivering better justice by replacing judges by algorithms).

It is clearly hard to expect a fully predictive theory of algorithmic design that would yield the ideal, *instance optimal* performance. However, the growing ease with which one can completely neglect any theoretical analysis and resign oneself to the output given by general heuristics (such as deep networks) can be dangerous. My personal feeling is that there is plenty more theoretical understanding to be gained about modeling classes of real-world data, designing general algorithmic techniques to efficiently and accurately solve them, and evaluating the performance of heuristics. This challenge is central to the field and to society. I expect that the growing interactions with the natural sciences will add both new questions and new ideas to this study.

---

[30]Including average-case analysis [Lev86], which focuses on hardness rather than easiness.

20.7.3   Resting cryptography on stronger foundations

Cryptography is discussed at length in Chapter 18. Crypto is by far the most extensive and repeatedly surprising study of the remarkable consequences of intractability assumptions. Furthermore, these consequences (and thus, the assumptions needed for them) form the current foundations of computer privacy and security and e-commerce, which are used by individuals, companies, armies, and governments. One would imagine that society would test these foundations and prepare for potential fragility at least as well as it does for, say, nuclear reactor accidents, earthquakes, and other potential sources of colossal disasters.

However, as discussed in Section 18.11, although most of cryptography requires the existence of *trap-door* functions, we have precious few candidates for these, of which integer factoring is still the main one (other important candidates are based on elliptic curves, lattices, and noisy linear equations). Moreover, short of the fact that these problems have not been found to be easily solvable yet, we have no shred of evidence that any of them is in fact difficult. For example, an efficient algorithm for any of these few problems will not imply efficient algorithms for a large collection of many other problems that are believed to be hard, and it will not imply the collapse of any complexity classes.

I find this state of affairs quite worrisome and feel that far more effort should be devoted to the challenge of finding more secure foundations for cryptography. The world has too much to lose if these foundations collapse. And it should be stressed that cryptographic research is thriving, mainly expanding the frontier of applications and consequences of even stronger (so, less believable) intractability assumptions than trap-door functions.[31] Naturally, applications are often first discovered under strong or specific assumptions, which are later weakened. Still, caution is necessary, certainly when resting the security of actual systems on shaky assumptions.

Of course, part of this state of affairs (namely of having few reliable, useful assumptions) is not for lack of trying: This challenge is indeed formidable. Given the first challenge of proving super-linear lower bounds for, well, anything, we can't expect yet super-polynomial lower bounds for factoring, for example. One natural direction is simply developing more candidates for trap-door functions, in case others break. Here the difficulty seems to be the (often algebraic) structure these problems seem to "demand." Another is developing reductions between trap-door functions to show that efficient algorithms for any of them have nontrivial consequences. Here again the rigid structure seems to resist reductions. Finally, resting cryptography on more believable intractability, preferably $\mathcal{P} \neq \mathcal{NP}$ (or at least its average-case version) runs into impossibility results against black-box use of such assumptions. Barak [Bar01] was the first to go beyond such black-box use; much more has followed. But it seems that further development of non-black-box crypto techniques (in which there definitely is exciting activity) is essential to this challenge.

20.7.4   Exploring physical reality vs. computational complexity

The celebrated Church-Turing thesis always struck me as an amazingly bold challenge. After all, it basically asserts:

Everything computable is computable by a Turing machine.

where "everything" is any well-defined function with discrete input and output domains. This thesis challenges anyone, scientists most prominently, to come up with such a well-defined computational task, which can be performed somehow in the real world (including the ability to prepare the input and read off the output), with any physical means whatsoever, but which cannot be computed by a Turing machine (one of the simplest physical devices imaginable). Still, in the 80 years that have passed, no one has come forth with a serious response to this challenge.

Soon after complexity theory was developed in the 1970s, people quickly extended this thesis to the far bolder thesis that it holds even if we restrict ourselves to *efficient* computation. This is sometimes called the "strong" or "feasible" Church-Turing thesis, and essentially states:

---

[31]The choice of proper cryptographic assumptions is itself a field of study in cryptography, discussed, for example, in [GK16] and its references.

Everything efficiently computable is efficiently computable by a Turing machine.

Here "efficient" is usually equated with "polynomial" (although one can adopt more stringent efficiency measures). This means that physical resources expended in the computational process (like energy and mass), as well as time and space, which are used in the computational process are allowed to grow only polynomially with the input size.

This thesis, together with the $\mathcal{P} \neq \mathcal{NP}$ conjecture, presents a seemingly much more tangible challenge: Simply find a natural process that can solve some $\mathcal{NP}$-complete problems with polynomial resources. And indeed, here came numerous suggestions, many (but certainly not all) explained and discussed in Aaronson's beautiful survey [Aar05]. As we discussed in Section 3.10, many of such proposals strengthen the case for adding $\mathcal{P} \neq \mathcal{NP}$ (and more generally, computational complexity considerations) as a guiding principle for modeling natural phenomena.[32]

However, here I want to turn around the challenge provided by the strong Church-Turing thesis, and consider it not only as a challenge to scientists to either refute it or use computational complexity in modeling natural processes, but rather as a challenge to complexity theorists to computationally abstract and model those (potential) gifts of nature that may eventually be used to enhance our laptops. Rising to this challenge (or rather challenges, as they arise from all walks of nature) has already been extremely important to foundational questions about efficient computation and algorithms, as well as to practice. Two of the main examples of exploring such gifts occupied several chapters in this book: The gift of randomness and the gift of "quantumness." I'll start with these and then mention a few others that are less developed.

First, nature seems to provide us with plenty of unpredictable phenomena, leading to the extremely important (theoretically and practically) theories of probabilistic algorithms, probabilistic proofs, pseudo-randomness, and randomness extraction. The first two theories postulate access to *perfect randomness*, namely, a sequence of unbiased, independent coin tosses, for the benefit of enhancing computational power. These extremely rich theories are discussed at length respectively in Chapters 7 and 10. The last two theories explore relaxing (in different ways) this strong postulate (of having access to perfect randomness), which may be violated in the real world. One (described in Section 7.2) explores having no access to randomness *at all* (and replaces it with a universally believed hardness assumption for "generating unpredictability"). The other (described in Chapter 9) assumes we can only access very weak random sources, for example, sequences of highly biased and correlated bits (and explains how to enhance their quality).

Now, practical use of "random" sequences in computer programs (e.g., in Monte Carlo simulations for weather prediction and nuclear interactions) predates all these theories, and moreover, it seemed that in many cases it did not matter how arbitrary (including deterministic!) was the "random" source that was used to yield "useful" results. But of course, there were no guarantees in these situations, nor understanding when such sources "worked" and when they did not. The two theories provide a formal basis for the study of randomness requirements of probabilistic algorithms, showing them provably applicable in settings for beyond the original requirement of perfect randomness. These theories justified and guided the choices of "random" sources to be used, identified sources that should not be used, and found efficient means to enhance weak ones. The de-randomization results (especially $\mathcal{BPP} = \mathcal{P}$) that follow from these theories impact the Church-Turing thesis directly: They justify replacing deterministic Turing machines with probabilistic ones (that err with small probability) as a legitimate definition of what efficient computation *is*.[33] Beyond all of that, as I discuss in this book, the abstract study of randomness has been incredibly useful to the theory and practice of computation (and mathematics) in many unexpected ways, far beyond the original intent.

Next, nature actually seems to provide us with quantum mechanical phenomena, of which spitting out perfectly random coin tosses is among the simplest powers. Indeed, this power seems to promise the possibility of efficiently manipulating (some) exponentially long pieces of information! This promise led to the theoretical study of using such quantum phenomena in computation, formulation of the quantum mechanical Turing machine, and the study of its power and limits (Chapter 11 is devoted to this exciting research area). Here theory is far from fully developed. Although we have formal universal models and equivalences between

---

[32]Which in many of these proposals also bring out the need to understand the actual inputs fed to these "heuristics of nature" in the sense discussed in the "heuristics challenge" in Section 20.7.2.

[33]This does not belittle the importance of using randomness, when it gives polynomial speed-ups.

them, important understanding of certain quantum (decoherence) error correction, quantum cryptographic protocols, and numerous interactions with physicists, we have only precious few problems that such a quantum computer can solve efficiently and a classical (probabilistic) one cannot.[34] Fortunately or not, these few problems include integer factoring and discrete logarithms, so this model undermines the current foundations of most cryptographic security systems. Of course, in this case (unlike with probabilistic computing), theory predated practice, and the algorithmic potential has led to the many, extremely challenging (and expensive) practical projects of building a quantum computer; literally attempting to add this quantum enhancement to our laptops and revising yet again our understanding of what an efficient Turing machine is.

The jury is out on which of the win-win outcomes will materialize: Either we will have such quantum laptops (significantly enhancing our computational abilities), or we will discover a fundamental gap in our models of nature that prevent their existence (and with it, hopefully revise the models and formulate better ones). But beyond motivating and assisting the practical project, the developed theories of quantum computing, proofs, advice, learning, games, and more have been extremely illuminating for the *classical* theory of computation in many unexpected ways (leading to lower bounds, quantum reductions between classical problems, no-signaling PCPs for delegation of computation, certification of randomness, and many others). Here, unlike with probabilistic computation, the belief is that quantum computation *does* add power to Turing machines (namely, that $\mathcal{BQP} \neq \mathcal{BPP}$). I find the discovery of more problems in the gap, as well as reasons despite it that $\mathcal{NP} \nsubseteq \mathcal{BQP}$ as well, to be an extremely challenging direction for computational complexity theory.

What else can we add to our laptops, or networks, or other algorithms, that will enhance the notion of what can be efficiently computed? There are many other gifts of nature, possibly more specific than the two above, which were already turned into computational models and paradigms whose algorithmic power and limits are being explored, and many others are being pursued. As discussed, I find that besides the natural scientific interest in such (and many other) models for the benefit of understanding nature, their purely theoretical pursuit as computational models will be beneficial for the broader understanding of computation itself, and perhaps will lead to the discovery of new algorithmic techniques, possibly resulting in the creation of new computational devices and systems. Many examples of natural phenomena with such potential are mentioned in this chapter. I thus look forward to the discovery of the power, limits, and relationships emanating from the study of such computational settings, formalized perhaps as DNA machines, nano machines, selfish networks, or natural algorithms (among many others), enriching both the practice of computing as well as its theory. Some of these may lead to classes like nano-$\mathcal{P}$, selfish-$\mathcal{P}$ and to possible probabilistic, quantum, nondeterministic, space-bounded, nonuniform, ..., variants, and to new connections to previously studied models that will deepen our understanding of computation.

20.8   K–12 education

This is a huge subject, so the discussion here is quite superficial and unfocused, which is aggravated by the lack of proper references (which may well exist). It can be regarded perhaps as a short wish-list, and a call-to-arms for the participation of ToC professionals in this effort.

ToC has so much to offer in making the education curriculum (at all levels) of every person so much better, richer, more valuable, and fun that it would take a separate book to discuss this. Of course, I am not original here, and there are plenty of thoughts and projects under way (with articles describing them) that suggest such efforts of integrating "computational thinking" into classrooms at various ages in various ways. As usual, even very good ideas have to eventually be implemented. As with other subjects, besides deciding what and how to teach kids, the most nontrivial task is figuring out how to train teachers to successfully deliver this material, with enthusiasm, to students. To make better decisions on both principles and implementation, I strongly hope that members of our community who are inclined to do so will be involved in these developments. This challenge is at least as important as exploring the scientific challenges of our field.

Below I list a small sample of truths I wish all kids would learn (at the appropriate age) in a way that

---

[34]Beyond the basic problem of simulating quantum systems.

will excite them and make them (via appropriate examples and demonstrations) remember them forever, no matter what life interests they end up pursuing. I believe this sampling captures examples of the kind of minimal knowledge and understanding about computation that all educated people should possess, in the same sense that we expect them to possess elementary knowledge and understanding about other topics, such as the basic sciences, history, and social systems.

- Turing, like Einstein, was one of the giants of twentieth-century science. His influence on technology and society was far greater than Einstein's.

- Computers cannot do everything. Indeed, there are basic, desirable computational tasks they can never accomplish. This is an absolute truth (a mathematical theorem), just like Pythagoras' theorem.

- What computers can do relies on the discovery of clever, efficient algorithms. One such algorithm can create a new industry, or predict and prevent an epidemic. The great discoverers of algorithms sit in the same hall of fame with great inventors, like Gutenberg, Edison, and Pasteur.

- Theories of nature should be predictive. Prediction is an algorithm! Predicting the weather or natural disasters *before* they happen requires these algorithms to be very efficient, *faster* than nature!

- Most math problems don't have only one right answer (with all others "wrong"). As in life, many mathematical tasks have solutions of different quality. Finding any solution is a good start, and trying to improve it is even better (this is discussed more below).

- The place value (e.g., decimal or binary) system that we all use to represent numbers was invented, and survived for millennia, only because it afforded extremely efficient storage and algorithms for performing arithmetic. Compare it to the unary system, or Roman numerals . . . .

- The method we learn to multiply numbers in grade school is first, an *algorithm*, and moreover, it is *not* the fastest algorithm known. There is a much faster one, which is still slower than the algorithm we have for adding numbers. It is a great intellectual mystery, with great practical implications, whether multiplication is inherently harder than addition.

- The world economy assumes computational hardness (at this time, this is the assumption that "inverting multiplication"—finding the prime factors of a large number—is an impossibly hard problem). Moreover, the world economy may collapse if this (or related) computational assumptions are false.

A bigger issue, in which I believe that ToC education has much to offer, is math education in elementary and high school. A lot has been written on the subject, and I will be brief on this point. For a more elaborate text, see this article by Fellows [Fel93], who has also invested much effort in developing classroom material for elementary schools and moreover has experimented with delivering it.

A recognized major malady of math education is the focus on arithmetic and algebra, with repetition ad nauseum of numerous, equivalent, boring numerical exercises to which the answer can be either right or wrong. In contrast, algorithmic problems on discrete objects offer, from kindergarten age on, an enormous variety of problems on, for example, games, puzzles, and mazes, which offer solutions of varied qualities and properties. For example, in optimization problems, every solution may be "correct," but their costs differ. Or, many different algorithms, strategies, and methods may solve a problem, but the amount of time or other resources they expend differ. In such problems, solutions of students invite the challenge *can you do better?* that is so much more constructive than *wrong!* Such problems also far better relate to the basic human experience all students feel when resolving life conflicts, from scheduling their activities to maximizing something (interest, pleasure, their parents' satisfaction spending their allowance, or playing better chess or soccer). Such problems also better relate to the above mentioned view of natural *processes* (themselves algorithms typically solving optimization problems), which should be highlighted in science courses, making a connection to math rather than through equations alone. Problems in which solutions of different qualities exist, and the quest to find better and better ones, allow students to develop deeper understanding, and, I believe, have much more fun. Moreover, such problems present math much better as

a *modeling* science, which allows precise formulation of such life and science situations, as well as of aspects like chance, uncertainty, partial information, and adversity. I feel that incorporating these ideas into math education, first for teachers and then for students, is very important (and I am not underestimating the difficulty of this task). This material should be part of every young person's knowledge in this information age, but at the same time, teaching it can also help transform the sad, negative attitude of many generations toward the very subject of mathematics. These ideas are not original and indeed have been attempted in many places. I am simply calling here for more participation of ToC professionals, to whom this way of thinking is so natural, in developing the materials and implementation details for this important endeavor.

20.9   The ToC community

I believe that the remarkable success of the field, partly summarized in this chapter, was not purely due to the intellectual content of ToC and the raw talents of its researchers, but also the way the ToC community organized and built itself. This section is devoted to this process and its character as I have experienced it, through very active participation in all relevant activities in all stages of my career. I allow myself generalities when the exceptions I know of are rare, fully realizing that my view is biased and that I tend to idealize.

Since the 1960s, much activity of ToC has been centered at and propelled by two yearly conferences, FOCS (Foundations of Computer Science) and STOC (Symposium on the Theory of Computing).[35] Taking place in the Fall and Spring, respectively, these forums became the meeting place (and the beating heart) of the whole community, a good fraction of which came to hear accounts of the most exciting developments in the field in the past 6 months. The presentations are determined by a program committee, which has no time to referee submissions but instead evaluates their novelty and impact on progress in the field. These committees have two effects. One is in the committee meetings themselves: 20 or so experts, of different research areas, opinions (and different ages!) meet periodically to discuss the submissions and make tough selection decisions, help clarify contributions, and shape directions. Second is the periodical meetings of a significant chunk of the ToC community, including many graduate students, at these conferences. The constant intellectual excitement about the challenges supplied in abundance by the field, and undoubtedly the personalities of its elders, have created a culture whose characteristics I'd like to list (and idealize). They undoubtedly play an important role in the success of the field so far, and it will be great to preserve and strengthen many of these as the growth and diversification of the field continue (which may be nontrivial). Needless to say, many of them are part of other academic disciplines, but I find their combination in ToC quite unique.

- *Level of discourse*. The need to convince a program committee, many members of which are not experts on your submission, has brought submissions to include long introductions with high-level motivation for the research described. Further, it has impacted the inclusion of intuitive explanations of technical developments. Restricted to being short, submissions therefore focus on ideas and concepts, which eases their propagation to nonexperts. The same effect shapes the lectures. As many members of the audience are nonexperts, motivation and intuition occupy significant parts of the short lectures. Thus, papers and lectures help highlight ideas and concepts, which can easily be understood by many. Moreover, this level of discussion has allowed the use and generalization of ideas and concepts developed in one context, possibly for a particular problem or application in completely different ones. This common high-level language, and the frequent exchange of the most recent important findings, has undoubtedly been responsible for the fast progress of the field. I believe that this also has an impact on the way ToC faculty teach their courses and write their books and surveys, highlighting motivation, ideas, concepts, and intuition before and beyond technical definitions and proofs.

- *Openness and evolution*. The horn of plenty that computation is, the diversity of its sources and manifestations, together with the varied interests of its researchers, has generated completely novel

---

[35] These conferences generally take place in North America, and I have attended most of them since 1980. Other conferences that encompass most areas in ToC, with some biases or different geographic locations were later added with a similar broad appeal, including ICALP, MFCS, SODA, and ITCS. What I say here is relevant to them as well.

models, questions, and research directions. These new interests compete for the attention of the community on making progress on existing and more established directions and open problems. It fell on the same bodies—the program committees of these conferences—to become the arbiters of the balance of new and old, and which parts of each to select for acceptance and presentations. These frequent discussions and debates, among researchers of different ages and tastes, made more valuable by the common language, not only inform the actual participants about the broad activities in the fields and their colleagues' opinions on them, but also has created a forum that repeatedly has to make tough choices on which of these trends will be visible and in what proportions. Looking back at the historical evolution of trends and directions, one can observe a great agility to pursue and investigate, as well as great openness to embrace and advertise completely new ideas, models, connections, and problems (emanating from technological, scientific or purely abstract considerations). This has created a wonderful dynamic, giving birth to many new research directions that were incubated at these forums. Many of these (e.g., most chapter titles in this book are perfect examples) grew sufficiently large to establish their own specialized conferences and communities, often operating in the same style. Of course, some other research directions have shrunk, died out, or merged with others.

- *A common core*. Despite this dynamic of topics entering and leaving this spotlight, central themes, mainly in the core algorithmic and complexity-theoretic areas of the field have maintained a long-term presence, even as they have grown. Many fundamental principles (of broad applicability across ToC) came from studying purely mathematical, *seemingly* "unmotivated" models and problems, which seemed "impractical" or "unnatural" generalizations of existing ones, or just came out of left field. In short, we followed the internal logic and aesthetics of the field, as is common in mathematics but much less so in application-driven areas. The openness of the community to such ideas paid back big time, as exemplified in many chapters in this book, and kept reestablishing the value of the core. Another key factor that made possible, as well as strengthened, the core was maintaining the level and style of discourse described above, despite the tremendous expansion in the diversity of areas covered. Naturally, this diversity necessitated many changes in the structure of the conferences and the sizes of program committees, but it still allowed the exchange of ideas, concepts, and methods across and between these very differently motivated and structured research directions. Moreover, the rate, fluidity, and quality of this exchange has clarified to what extent ToC is a cohesive field, and how important it is to have a core and an infrastructure to utilize it. In many cases, this flow of ideas, methods, and results was extremely natural, whereas in many others it required remarkable ingenuity and of course knowledge of what was going on in other areas. In a lecture on the value of such knowledge, I dubbed this phenomenon *depth through breadth*, the value and strength of which I find unique to our field. Some of the commonality of these many threads is captured in the methodology Section 20.3.

- *Community and attraction of talent*. Challenging and important long-term goals; a rich tapestry of concepts, ideas, and tools with exciting progress in connecting and expanding them; the building of theories to gradually access such understanding; and many other intellectual facets of a scientific field are necessary for continuously attracting young talent in the field who will further this work. And indeed, the theory of computation has all that and more, as this chapter and indeed the whole book show. But I would argue that the social and educational structure of the field makes it even more attractive for young talent to enter, stay, grow, and hopefully also discover, educate, and lead. I have already explained how the level of discourse makes it easy and inviting to newcomers, especially those exposed to it in undergraduate and grad schools, but also for researchers of different fields who are interested. Another important aspect is the "democracy" and social welcoming of the field, which generally considers raw ideas and not the seniority of those generating them. One of my formative experiences, as a first-year graduate student, was being introduced in the FOCS conference of 1980 to Richard Karp, *the* most eminent leader of the field. It was "Dick, this is Avi, Avi, this is Dick," which was followed by five or ten minutes in which Karp, with genuine interest, explored what areas of theory I like and might be pursuing. I quickly learned that no introductions were even needed, and

that leaders of the field were equally accessible when you came to them with a problem or an idea. I believe that this is the experience felt to this day by newcomers to the field, and that it has created a community of much collegiality, collaboration, and friendship. This atmosphere persists despite the natural and much welcome competitiveness and drive of ToC researchers.

Critique

After these many positive aspects, I think it makes sense to mention a couple of issues where I feel that our community has behaved suboptimally, in the hope that these will improve over time. They are all related, and probably stem from a single root cause, namely, an internal focus on the intellectual problems of the field, neglecting to some extent a necessary investment in the external spheres in which we live and function. By far the most significant neglect is in communicating with the outside world about ToC achievements, both their intellectual contents and their practical impact. Of course, some ToC members have invested in popular lectures, surveys, and books for the general computer science community, the general scientific community, and even the public over the years. But I fear that what was achieved by them is far smaller than what could have been achieved with much more (well justified) effort. There are many ways to demonstrate the general low level of understanding, acknowledgment, and appreciation of the amazing body of work created by ToC, outside the field. This state of affairs could easily improve, as many of our discoveries have such appealing conceptual contents and so are easy to relate to almost any audience. In short, I feel that the field should make it a priority, and hope that much more will be done on this important front.

One way of making it easier to communicate a topic, especially for the general public (but also for people outside one's domain) is having an evocative vocabulary to describe central notions and results. By far the field that excels over all others in this domain is physics. Of course, it starts with a huge natural advantage over most other fields, as their objects of study have aroused our curiosity and fascination from a very early age, like the stars, rainbows, ocean waves, flying birds, and so many more. But even for objects of study to which we have no direct access, physicists pick intriguing and captivating names like "big bang," "black holes," "dark matter," "supernovas," and call some properties of elementary particles "strange" and "charm." In contrast, not only does ToC start with no one having any computational or algorithmic experience at a formative age, we call our main concepts by boring, inexplicable acronyms like P, NP, BPP, SC, NC (and these are just a few of the most famous animals in the "complexity zoo").[36] There are numerous other choices made that I believe could be much better, and perhaps the best example is "complexity" itself (even in the phrase "computational complexity"), which is far too general and is constantly confused with other uses and meanings of this word. Much better phrases for what we are actually studying would have been "computational efficiency" and "computational intractability." Of course, some choices are excellent in being both meaningful and intriguing, like the concepts "zero-knowledge proofs," "Monte Carlo algorithms," "randomness extractors," "price of anarchy," and the problems "dining philosophers," "Byzantine generals," "perfect matching," and "maximum clique." I would argue that bad naming choices hinder our internal education of undergraduate and graduate ToC students, and not just our ability to communicate our work outside the field! Although some of the bad choices are too engrained and thus possibly too late to change, I feel that being conscious of this issue may give birth to better names of new concepts, and inspire inventive names for old concepts that the community can adopt in popular lectures (where using professional jargon is not necessary).

20.10   Conclusions

Scientific disciplines are born (and transformed) in different ways; volumes have been written on the subject. In the case of ToC, it is remarkably easy to determine the precise moment of birth. It happened in a singular event, following an age-old process. Computation, in its manifold forms, has been a subject of inquiry since the dawn of humanity, simply because the efficient utilization of resources is second (or first) nature to people (and, to all life forms, and indeed to inanimate matter subject to the laws of physics). But this inquiry was

---

[36] A website of most complexity classes is maintained at [AGKR17].

missing a formalism that would make it precise. This formalism was supplied in 1936 by Turing's seminal paper, which gave us the *Turing machine* model. It struck gold, finally enabling the creation of a theory! Computation revealed itself as a bottomless goldmine, and what has already been uncovered by ToC research stands as one of the greatest achievements in scientific history.

In this chapter, I discussed many facets of ToC, especially its past and potential contributions to human inquiry and to society. Its purly academic, mathematical pursuit probes some of the deepest intellectual questions asked by humanity. And as a bonus, it is of fundamental importance to most other sciences, and to technology and human life. In short, this chapter, and the book containing it, present the clear case that ToC is an independent, important academic discipline. I feel that this view should be the basis of any discussion (internal or external) regarding the field and its future.

This position of ToC in the intellectual arena comes with great responsibility. To continue to develop both internally and in the expanding and dynamic collaborations with other disciplines, I believe that the ToC community needs to grow, and such growth presents new challenges. Some of these will be adequately met by the spirit and social culture of this young field, which I hope will be preserved. Others have to do with the changing nature of the field and its increased diversity, and I feel require significant effort concentrated on education and on cohesion. Let me elaborate a bit.

ToC is in the beginning phase of what I expect will be a long endeavor. Like other centuries-old disciplines, such as mathematics, physics, biology, and philosophy, ToC may change form and focus many times in the future. I find that the current period is definitely one of such change, possibly a phase transition, due to the explosion of connections of the field to other scientific disciplines and the ever-growing demands for theory from the even larger explosion of computer science and engineering, generating new technologies and applications of computing systems at an unbelievable pace. These present enormous challenges, as both the nature and size of ToC are changing. I find a couple of (related) predictions inevitable, and each calls for action on the part of the ToC community. First, as the need for computational modeling and algorithm design throughout the sciences and industry expands, many more professionals will have to be trained in its principles and methodology. These principles will become prerequisite in most undergraduate education, and I strongly feel that ToC must play an active role in the design of such courses and curricula. Second, computational theory research and researchers will become integral parts of disciplines outside computer science, and at the same time, ToC research itself (which started as a pure math pursuit of computer system—motivated problems) will need to adapt and adopt experimental aspects of scientific and social theories studying the real world.

This great expansion and diversity, not only of research topics but also of methods and culture, is certainly a blessing, and indeed part of the calling of ToC. But it also means that maintaining a viable, central core of the field will become much harder. Hard as it is, however, I strongly believe that preserving such a core— which will allow the flow and exchange of fundamental ideas, techniques, and insights about computation relevant across disciplines–will be at least as important as it has been in the past. Conversely, I fear that fragmentation of the field can lose this important advantage! The challenge of preserving the cohesion of ToC is certainly nontrivial and will require thoughtful reorganization and adaptation. Besides creating physical and on-line forums to maintain this cohesion and exchange of ideas, I find that the field can greatly benefit from administrative changes in undergraduate education. Specifically, the creation of majors, certificates, and other undergraduate programs dedicated to ToC is natural. There is a wealth of material regarding foundations and connections of the field, and it is ripe for creating highly rich and challenging curricula. *The value, to both academia and industry, of a student graduating with this knowledge and understanding of theory, is extremely high.* Looking further, I feel that given the growing intellectual centrality of ToC to so many academic departments (and even though computer science will probably remain the strongest connection), it may become natural and beneficial, for universities and the field itself, to create *separate* ToC departments; this will reflect and reinforce both the field's independence and its centrality to others.

Building and maintaining the necessary structures, like those suggested here, to preserve cohesion of ToC, can only exist if the community is confident in its independence, value, and mission. Namely, it can happen only if ToC members fully appreciate the meaning, importance, scope, and impact of past achievements and the even greater potential of future ones. These conceptual messages should be an essential part of

the education we provide to our students (and to each other), well beyond the technical know-how. Such a foundation will hopefully facilitate further constructive discussion on the organization of the field, which I find so important for its future.

References

[AA11]   S. Aaronson and A. Arkhipov. The computational complexity of linear optics. In *Proceedings of the 43rd annual ACM symposium on Theory of Computing*, pages 333–342. ACM, 2011. Cited at 116

[Aar03]   S. Aaronson. Is "P versus NP" formally independent? *Bulletin of the EATCS*, 81:109–136, 2003. Cited at 55

[Aar05]   S. Aaronson. Guest column: NP-complete problems and physical reality. *ACM SIGACT News*, 36(1):30–52, 2005. Cited at 35, 261

[Aar13a]   S. Aaronson. *Quantum computing since Democritus*. Cambridge University Press, 2013. Cited at 114

[Aar13b]   S. Aaronson. Why philosophers should care about computational complexity. *Computability: Turing, Gödel, Church, and beyond*, pages 261–328, 2013. Cited at 253

[Aar16a]   S. Aaronson. The complexity of quantum states and transformations: From quantum money to black holes. *arXiv preprint arXiv:1607.05256*, 2016. Cited at 248

[Aar16b]   Scott Aaronson. P=?NP. In *Open problems in mathematics*, pages 1–122. Springer, 2016. Cited at 26

[AAVL11]   D. Aharonov, I. Arad, U. Vazirani, and Z. Landau. The detectability lemma and its applications to quantum Hamiltonian complexity. *New Journal of Physics*, 13(11):113043, 2011. Cited at 119, 121

[AB87]   N. Alon and Boppana R. B. The monotone circuit complexity of Boolean functions. *Combinatorica*, 7(1):1–22, 1987. Cited at 53

[AB03]   M. Agrawal and S. Biswas. Primality and identity testing via Chinese remaindering. *Journal of the ACM*, 50(4):429–443, 2003. Cited at 87, 136

[AB09]   S. Arora and B. Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009. Cited at 5

[ABBG11]   S. Arora, B. Barak, M. Brunnermeier, and R. Ge. Computational complexity and information asymmetry in financial products. *Communications of the ACM*, 54(5):101–107, 2011. Cited at 252

[ABF+08]   M. Alekhnovich, M. Braverman, V. Feldman, A.R. Klivans, and T. Pitassi. The complexity of properly learning simple concept classes. *Journal of computer and system sciences*, 74(1):16–34, 2008. Cited at 200

[ABL02]   S. Arora, B. Bollobás, and L. Lovász. Proving integrality gaps without knowing the linear program. In *Proceedings of 43rd annual IEEE Symposium on Foundations of Computer Science*, pages 313–322. IEEE, 2002. Cited at 235

[ABND+87]   H. Attiya, A. Bar-Noy, D. Dolev, D. Koller, D. Peleg, and R. Reischuk. Achievable cases in an asynchronous environment. In *Proceedings of 28th annual IEEE Symposium on Foundations of Computer Science*, pages 337–346, 1987. Cited at 220, 225

[ABO97]   D. Aharonov and M. Ben-Or. Fault-tolerant quantum computation with constant error. In *Proceedings of the 29th annual ACM symposium on Theory of Computing*, pages 176–188. ACM, 1997. Cited at 117

[ABOE10]   D. Aharonov, M. Ben-Or, and E. Eban. Interactive proofs for quantum computation. In *Proceedings of innovations of Computer Science (ICS 2010), China*, pages 453–469, 2010. Cited at 121

[ABOEM17] D. Aharonov, M. Ben-Or, E. Eban, and U. Mahadev. Interactive proofs for quantum computations. *arXiv preprint arXiv:1704.04487*, 2017. Cited at 121

[AC11]   H. Attiya and A. Castañeda. A non-topological proof for the impossibility of *k*-set agreement. In *Symposium on self-stabilizing systems*, pages 108–119. Springer, 2011. Cited at 226

[ACORT11] D. Achlioptas, A. Coja-Oghlan, and F. Ricci-Tersenghi. On the solution-space geometry of random constraint satisfaction problems. *Random Structures & Algorithms*, 38(3):251–268, 2011. Cited at 144

[AD97]   M. Ajtai and C. Dwork. A public-key cryptosystem with worst-case/average-case equivalence. In *Proceedings of the 29th annual ACM symposium on Theory of Computing*, pages 284–293. ACM, 1997. Cited at 47, 148

[Adl78]   L. Adleman. Two theorems about random polynomial time. In *Proceedings of 19th annual IEEE Symposium on Foundations of Computer Science*, pages 75–83. IEEE, 1978. Cited at 71

[Adl94]   L.M. Adleman. Molecular computation of solutions to combinatorial problems. *Nature*, 369:40, 1994. Cited at 244

[AEH75]   E.A. Akkoyunlu, K. Ekanadham, and R.V. Huber. Some constraints and tradeoffs in the design of network communications. In *ACM SIGOPS operating systems review*, volume 9, pages 67–74. ACM, 1975. Cited at 223

[AFG14]   A. Ambainis, Y. Filmus, and F. Gall. Fast matrix multiplication: Limitations of the laser method. *arXiv preprint arXiv:1411.5414*, 2014. Cited at 128

[AFR06]   J. Aspnes, F.E. Fich, and E. Ruppert. Relationships between broadcast and shared memory in reliable anonymous distributed systems. *Distributed Computing*, 18(3):209–219, 2006. Cited at 220

[AFT11]   B. Alexeev, M.A. Forbes, and J. Tsimerman. Tensor rank: Some lower and upper bounds. In *2011 IEEE 26th annual conference on Computational Complexity (CCC)*, pages 283–291. IEEE, 2011. Cited at 133

[AGHP92]   N. Alon, O. Goldreich, J. Håstad, and R. Peralta. Simple constructions of almost *k*-wise independent random variables. *Random Structures & Algorithms*, 3(3):289–304, 1992. Addendum: random structures and algorithms 4. Cited at 85

[AGKR17]   S. Aaronson, C. Granade, G. Kuperberg, and V. Russo. The complexity zoo, 2017. https://complexityzoo.uwaterloo.ca/Complexity_Zoo. Cited at 266

[AH89]   K.I. Appel and W. Haken. *Every planar map is four colorable*, volume 98. American Mathematical Society, 1989. Cited at 14

[AH92]   L.M. Adleman and M.A. Huang. *Primality testing and Abelian varieties over finite fields*, volume 1512. Springer, 1992. Cited at 136

[AH17]   E. Allender and S. Hirahara. New insights on the (non-)hardness of circuit minimization and related problems. In *Mathematical Foundations of Computer Science*, volume 83 of *LIPIcs*, pages 54:1–54:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. Cited at 37, 40, 56

[AHK12]   S. Arora, E. Hazan, and S. Kale. The multiplicative weights update method: A meta-algorithm and applications. *Theory of Computing*, 8(1):121–164, 2012. Cited at 95, 182, 235

[AHT06]   I. Agol, J. Hass, and W. P. Thurston. The computational complexity of knot genus and spanning area. *Transactions of the American Mathematical Society*, 358:3821–3850, 2006. Cited at 32

[Ajt83]   M. Ajtai. σ_1-formulae on finite structures. *Annals of the Pure and Applied Logic*, 24(1):1–48, 1983. Cited at 52, 55

[Ajt96]   M. Ajtai. Generating hard instances of lattice problems. In *Proceedings of the 28th annual ACM symposium on Theory of Computing*, pages 99–108. ACM, 1996. Cited at 47, 148

[AK09]   N. Alon and B. Klartag. Economical toric spines via Cheeger's inequality. *Journal of Topology and Analysis*, 1(02):101–111, 2009. Cited at 147

[AKK17]   P. Austrin, P. Kaski, and K. Kubjas. Tensor network complexity of multilinear maps. *arXiv preprint arXiv:1712.09630*, 2017. Cited at 119

[AKN98]   D. Aharonov, A. Kitaev, and N. Nisan. Quantum circuits with mixed states. In *Proceedings of the 30th annual ACM symposium on Theory of Computing*, pages 20–30. ACM, 1998. Cited at 114

[AKS83]   M. Ajtai, J. Komlós, and E. Szemerédi. An O(n log n) sorting network. In *Proceedings of the 15th annual ACM symposium on Theory of Computing*, pages 1–9. ACM, 1983. Cited at 55

[AKS87]   M. Ajtai, J. Komlós, and E. Szemerédi. Deterministic simulation in LOGSPACE. In *Proceedings of the 19th annual ACM symposium on Theory of Computing*, pages 132–140. ACM, 1987. Cited at 101

[AKS04]   M. Agrawal, N. Kayal, and N. Saxena. Primes is in *P*. *Annals of Mathematics*, 160(2):781–793, 2004. Cited at 19, 28, 80, 87, 136

[AL88]   D. Angluin and P. Laird. Learning from noisy examples. *Machine Learning*, 2(4):343–370, 1988. Cited at 192

[AL11]   G. Asharov and Y. Lindell. A full proof of the BGW protocol for perfectly secure multiparty computation. *Journal of Cryptology*, 30:1–94, 2011. Cited at 214

[Ald83]   D. J. Aldous. Random walks on finite groups and rapidly mixing Markov chains. In *Séminaire de Probabilités XVII 1981/82*, pages 243–297. Springer, 1983. Cited at 142

[Ald90]   D. J. Aldous. The random walk construction of uniform spanning trees and uniform labelled trees. *SIAM Journal on Discrete Mathematics*, 3(4):450–465, 1990. Cited at 143

[ALGV18]   N. Anari, K. Liu, S. O. Gharan, and C. Vinzant. Log-concave polynomials II: High-dimensional walks and an FPRAS for counting bases of a matroid. *arXiv preprint arXiv:1811.01816*, 2018. Cited at 144

[All17]   E. Allender. The complexity of complexity. In *Computability and Complexity*, pages 79–94. Springer, 2017. Cited at 40

[ALM+98]   S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and the hardness of approximation problems. *Journal of the ACM*, 45(3):501–555, 1998. Cited at 110, 235

[ALN08]   S. Arora, J. Lee, and A. Naor. Euclidean distortion and the sparsest cut. *Journal of the American Mathematical Society*, 21(1):1–21, 2008. Cited at 140

[Alo86]   N. Alon. Eigenvalues and expanders. *Combinatorica*, 6(2):83–96, 1986. Cited at 90, 91

[ALW01]   N. Alon, A. Lubotzky, and A. Wigderson. Semi-direct product in groups and zig-zag product in graphs: Connections and applications. In *Proceedings of 42nd annual IEEE Symposium on Foundations of Computer Science*, pages 630–637. IEEE, 2001. Cited at 92

[AM75]   L. Adleman and K. Manders. Computational complexity of decision problems for polynomials. In *Proceedings of 16th annual IEEE Symposium on Foundations of Computer Science*, pages 169–177. IEEE, 1975. Cited at 32

[AM85]   N. Alon and V. D. Milman. λ_1, isoperimetric inequalities for graphs, and superconcentrators. *Journal of Combinatorial Theory*, 38(1):73–88, 1985. Cited at 90

[AM88]   N. Alon and W. Maass. Meanders and their applications in lower bounds arguments. *Journal of computer and system sciences*, 37(2):118–129, 1988. Cited at 165

[AMPS13]   A. Almheiri, D. Marolf, J. Polchinski, and J. Sully. Black holes: Complementarity or firewalls? *Journal of High Energy Physics*, 2013(2):62, 2013. Cited at 248

[AMS96]   N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. In *Proceedings of the 28th annual ACM symposium on Theory of Computing*, pages 20–29. ACM, 1996. Cited at 157

[And87]   A.E. Andreev. About one method of obtaining more than quadratic effective lower bounds of complexity of PI-schemes. *Vestnik Moskovskogo Universiteta Seriya 1 Matematika Mekhanika*, (1):70–73, 1987. Cited at 52

[Ank52]   N.C. Ankeny. The least quadratic non residue. *Annals of Mathematics*, pages 65–72, 1952. Cited at 87

[ANS10]   B. Adsul, S. Nayak, and K.V. Subrahmanyam. A geometric approach to the Kronecker problem ii: Invariants of matrices for simultaneous left-right actions. *Manuscript, available in http://www.cmi.ac.in/kv/ANS10.pdf*, 18, 2010. Cited at 152

[APR83]   L.M. Adleman, C. Pomerance, and R.S. Rumely. On distinguishing prime numbers from composite numbers. *Annals of Mathematics*, pages 173–206, 1983. Cited at 136

[AR05]   D. Aharonov and O. Regev. Lattice problems in NP $\cap$ coNP. *Journal of the ACM*, 52(5):749–765, 2005. Cited at 40

[AR08]   M. Alekhnovich and A. Razborov. Resolution is not automatizable unless w[p] is tractable. *SIAM Journal on Computing*, 38(4):1347–1363, 2008. Cited at 68

[Arı09]   E. Arıkan. Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels. *IEEE Transactions on Information Theory*, 55(7):3051–3073, 2009. Cited at 176

[Aro94]   S. Arora. *Probabilistic checking of proofs and the hardness of approximation problems*. PhD thesis, UC Berkeley, 1994. Revised version at `http://eccc.hpi-web.de/eccc-local/ECCC-Books/sanjeev_book_readme.html`. Cited at 110

[Art27]   E. Artin. Über die zerlegung definiter funktionen in quadrate. In *Abhandlungen aus dem mathematischen Seminar der Universität Hamburg*, volume 5, pages 100–115. Springer, 1927. Cited at 65

[AS83]   D. Angluin and C.H. Smith. Inductive inference: Theory and methods. *ACM Computing Surveys (CSUR)*, 15(3):237–269, 1983. Cited at 190

[AS98]   S. Arora and S. Safra. Probabilistic checking of proofs: A new characterization of NP. *Journal of the ACM*, 45(1):70–122, 1998. Cited at 235

[AS00]   N. Alon and J. Spencer. *The probabilistic method*. Wiley-Interscience Series on Discrete Mathematics and Optimization. John Wiley & Sons, 2nd edition, 2000. Cited at 55, 82, 103

[AS03]   S. Arora and M. Sudan. Improved low-degree testing and its applications. *Combinatorica*, 23(3):365–426, 2003. Cited at 236

[AS10]   V. Arvind and S. Srinivasan. On the hardness of the noncommutative determinant. In *Proceedings of the 42nd annual ACM symposium on Theory of Computing*, pages 677–686. ACM, 2010. Cited at 134

[Asp99]   A. Aspect. Bell's inequality test: More ideal than ever. *Nature*, 398(6724):189–190, 1999. Cited at 121

[Asp03]   J. Aspnes. Randomized protocols for asynchronous consensus. *Distributed Computing*, 16(2–3):165–175, 2003. Cited at 224

[AV08]   M. Agrawal and V. Vinay. Arithmetic circuits: A chasm at depth four. In *Proceedings of 49th annual IEEE Symposium on Foundations of Computer Science*, pages 67–75. IEEE, 2008. Cited at 133

[AvDK⁺08]   D. Aharonov, W. van Dam, J. Kempe, Z. Landau, S. Lloyd, and O. Regev. Adiabatic quantum computation is equivalent to standard quantum computation. *SIAM Review*, 50(4):755–787, 2008. Cited at 120

[AW85]   M. Ajtai and A. Wigderson. Deterministic simulation of probabilistic constant depth circuits. In *Proceedings of 26th annual IEEE Symposium on Foundations of Computer Science*, pages 11–19. IEEE, 1985. Cited at 79

[AW04]   H. Attiya and J. Welch. *Distributed computing: Fundamentals, simulations, and advanced topics*, volume 19. John Wiley & Sons, 2004. Cited at 218, 219

[AW09]   S. Aaronson and A. Wigderson. Algebrization: A new barrier in complexity theory. *ACM Transactions on Computation Theory (TOCT)*, 1(1):2, 2009. Cited at 48, 56

[AW18]   J. Alman and V.V. Williams. Limits on all known (and some unknown) approaches to matrix multiplication. In *Proceedings of 59th annual IEEE Symposium on Foundations of Computer Science*, pages 580–591. IEEE, 2018. Cited at 56, 128

[AZGMS14]   Z. Allen-Zhu, R. Gelashvili, S. Micali, and N. Shavit. Sparse sign-consistent Johnson-Lindenstrauss matrices: Compression with neuroscience-based constraints. *Proceedings of the National Academy of Sciences*, 111(47):16872–16876, 2014. Cited at 247

[AZH16]   Z. Allen-Zhu and E. Hazan. Variance reduction for faster non-convex optimization. In *International conference on Machine Learning*, pages 699–707, 2016. Cited at 235

[AZO14]   Z. Allen-Zhu and L. Orecchia. Linear coupling: An ultimate unification of gradient and mirror descent. *arXiv preprint arXiv:1407.1537*, 2014. Cited at 235

[Bab85]   L. Babai. Trading group theory for randomness. In *Proceedings of the 17th annual ACM symposium on Theory of Computing*, symposium on the Theory of Computing '85, pages 421–429, New York, NY, USA, 1985. ACM. Cited at 104

[Bab90]   L. Babai. E-mail and the unexpected power of interaction. *Proceedings of the 5th annual conference on structure in complexity theory*, pages 30–44, 1990. Cited at 105

[Bab91]   L. Babai. Local expansion of vertex-transitive graphs and random generation in finite groups. In *Proceedings of the 23rd annual ACM symposium on Theory of Computing*, volume 91, pages 164–174. Citeseer, 1991. Cited at 142

[Bab15]   L. Babai. Graph isomorphism in quasipolynomial time. *arXiv preprint arXiv:1512.03547*, 2015. Cited at 20, 40, 103, 141

[Bar86]   D.A. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. In *Proceedings of the 18th annual ACM symposium on Theory of Computing*, pages 1–5. ACM, 1986. Cited at 159, 241

[Bar01]   B. Barak. How to go beyond the black-box simulation barrier. In *Proceedings of 42nd annual IEEE Symposium on Foundations of Computer Science*, pages 106–115. IEEE, 2001. Cited at 208, 260

[Bar04]   B. Barak. Non-black-box techniques in cryptography. *PhD Dissertation, The Weizmann Institute of Science, Computer Science department*, 2004. Cited at 112

[Bar16]   A. Barvinok. *Combinatorics and complexity of partition functions*. Springer, 2016. Cited at 237

[BB84]   C.H. Bennett and G. Brassard. Quantum cryptography: Public key distribution and coin tossing. In *Proceedings of IEEE international conference on computers, systems and signal processing*, volume 175. New York, 1984. Cited at 118

[BBC⁺93]   C.H. Bennett, G. Brassard, C. Crépeau, R. Jozsa, A. Peres, and W.K. Wootters. Teleporting an unknown quantum state via dual classical and Einstein-Podolsky-Rosen channels. *Physical Review Letters*, 70(13):1895, 1993. Cited at 118

[BBCR13]   B. Barak, M. Braverman, X. Chen, and A. Rao. How to compress interactive communication. *SIAM Journal on computing*, 42(3):1327–1363, 2013. Cited at 172, 173, 175

[BBG13]   M. Balcan, A. Blum, and A. Gupta. Clustering under approximation stability. *Journal of the ACM*, 60(2):8, 2013. Cited at 43, 259

[BC06]   M. Braverman and S. Cook. Computing over the reals: Foundations for scientific computing. *Notices of the AMS*, 53(3):318–329, 2006. Cited at 16

[BCC⁺17]   J. Blasiak, T. Church, H. Cohn, J.A. Grochow, and C. Umans. Which groups are amenable to proving exponent two for matrix multiplication? *arXiv preprint arXiv:1712.02302*, 2017. Cited at 128

[BCE⁺95]   P. Beame, S. Cook, J. Edmonds, R. Impagliazzo, and T. Pitassi. The relative complexity of NP search problems. In *Proceedings of the 27th annual ACM symposium on Theory of Computing*, pages 303–314. ACM, 1995. Cited at 38

[BCHP17]   S. Benoit, J. Colliard, C. Hurlin, and C. Pérignon. Where the risks lie: A survey on systemic risk. *Review of Finance*, 21(1):109–152, 2017. Cited at 252

[BCL⁺18]   S. Bubeck, M. B. Cohen, Y. T. Lee, J. R. Lee, and A. Madry. *k*-server via multiscale entropic regularization. In *Proceedings of the 50th Annual ACM Symposium on Theory of Computing*, pages 3–16. ACM, 2018. Cited at 181

[BCS10]   P. Bürgisser, M. Clausen, and M.A. Shokrollahi. *Algebraic Complexity Theory*. Springer Publishing Company, Incorporated, 2010. Cited at 124

[BCSS98]   L. Blum, F. Cucker, M. Shub, and S. Smale. *Complexity and Real Computation*. Springer-Verlag, 1998. Cited at 16

[BD02]   A. Blum and J. Dunagan. Smoothed analysis of the perceptron algorithm for linear programming. In *Proceedings of the 13th annual ACM-SIAM symposium on discrete algorithms*, pages 905–914. SIAM, 2002. Cited at 187

[BDBK+94]   S. Ben-David, A. Borodin, R. Karp, G. Tardos, and A. Wigderson. On the power of randomization in on-line algorithms. *Algorithmica*, 11(1):2–14, 1994. Cited at 179, 181

[BDSMP91]   M. Blum, A. De Santis, S. Micali, and G. Persiano. Noninteractive zero-knowledge. *SIAM Journal on Computing*, 20(6):1084–1118, 1991. Cited at 109

[BDWY13]   B. Barak, Z. Dvir, A. Wigderson, and A. Yehudayoff. Fractional Sylvester-Gallai theorems. *Proceedings of the National Academy of Sciences*, 110(48):19213–19219, 2013. Cited at 138

[Bea94]   P. Beame. A switching lemma primer. Technical report, Technical Report UW-CSE-95-07-01, Department of Computer Science and Engineering, University of Washington, 1994. Cited at 52

[Bea97]   R. Beals. Quantum computation of Fourier transforms over symmetric groups. In *Proceedings of the 29th annual ACM symposium on Theory of Computing*, pages 48–53. ACM, 1997. Cited at 116

[Bec91]   J. Beck. An algorithmic approach to the Lovász local lemma. i. *Random Structures & Algorithms*, 2(4):343–365, 1991. Cited at 144, 238

[BEHW90]   A. Blumer, A. Ehrenfeucht, D. Haussler, and M.K. Warmuth. Occam's razor. *Readings in machine learning*, pages 201–204, 1990. Cited at 196, 197

[Bel64]   J.S. Bell. On the Einstein-Podolsky-Rosen paradox. *Physics*, 1(3):195–200, 1964. Cited at 121

[Ben80]   P. Benioff. The computer as a physical system: A microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines. *Journal of Statistical Physics*, 22(5):563–591, 1980. Cited at 114, 242

[Ber67]   E.R. Berlekamp. Factoring polynomials over finite fields. *Bell System Technical Journal*, 46(8):1853–1859, 1967. Cited at 71, 137

[Ber84]   S.J. Berkowitz. On computing the determinant in small parallel time using a small number of processors. *Information processing letters*, 18(3):147–150, 1984. Cited at 129

[Bes19]   A. Besicovitch. Sur deux questions d'integrabilité des fonctions. *J. Soc. Phys. Math.*, 2:105–123, 1919. Cited at 137

[BEY05]   A. Borodin and R. El-Yaniv. *Online computation and competitive analysis.* Cambridge University Press, 2005. Cited at 179

[BF28]   M. Born and V. Fock. Beweis des adiabatensatzes. *Zeitschrift für Physik*, 51(3-4):165–180, 1928. Cited at 120

[BF90]   D. Beaver and J. Feigenbaum. Hiding instances in multioracle queries. *Annual Symposium on Theoretical Aspects of Computer Science*, pages 37–48, 1990. Cited at 106

[BFK09]   A. Broadbent, J. Fitzsimons, and E. Kashefi. Universal blind quantum computation. In *Proceedings of 50th annual IEEE Symposium on Foundations of Computer Science*, pages 517–526. IEEE, 2009. Cited at 121

[BFK10]   A. Broadbent, J. Fitzsimons, and E. Kashefi. QMIP= MIP*. *arXiv preprint arXiv:1004.1130*, 2010. Cited at 121

[BFL91]   L. Babai, L. Fortnow, and C. Lund. Non-deterministic exponential time has two-prover inter-active protocols. *Computational complexity*, 1(1):3–40, 1991. Cited at 106, 236

[BFS86]   L. Babai, P. Frankl, and J. Simon. Complexity classes in communication complexity theory. In *Proceedings of 27th annual IEEE Symposium on Foundations of Computer Science*, pages 337–347. IEEE, 1986. Cited at 161, 163

[BG93]   E. Borowsky and E. Gafni. Generalized FLP impossibility result for t-resilient asynchronous computations. In *Proceedings of the 25th annual ACM symposium on Theory of Computing*, pages 91–100. ACM, 1993. Cited at 225, 226

[BG08]   J. Bourgain and A. Gamburd. Uniform expansion bounds for Cayley graphs of $SL_2(\mathbb{F}_p)$. *Annals of Mathematics*, 167(2):625–642, 2008. Cited at 91, 92

[BG10]   J. Bourgain and A. Gamburd. Spectral gaps in $SU(d)$. *Comptes Rendus Mathematique*, 348(11):609–611, 2010. Cited at 93

[BGBD+04] Y. Benenson, B. Gil, U. Ben-Dor, R. Adar, and E. Shapiro. An autonomous molecular com-puter for logical control of gene expression. *Nature*, 429(6990):423–429, 2004. Cited at 244

[BGGT13]   E. Breuillard, B. Green, R. Guralnick, and T. Tao. Expansion in finite simple groups of Lie type. *arXiv preprint arXiv:1309.1975*, 2013. Cited at 93

[BGI+01]   B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan, and K. Yang. On the (im)possibility of obfuscating programs. In *Annual International Cryptology Conference*, pages 1–18. Springer, 2001. Cited at 216

[BGS75]   T. Baker, J. Gill, and R. Solovay. Relativizations of the P =? NP question. *SIAM Journal on Computing*, 4:431–442, 1975. Cited at 48

[BGS10]   J. Bourgain, A. Gamburd, and P. Sarnak. Affine linear sieve, expanders, and sum-product. *Inventiones mathematicae*, 179(3):559–644, 2010. Cited at 93

[BGT10]   E. Breuillard, B. Green, and T. Tao. Suzuki groups as expanders. *arXiv preprint arXiv:1005.0782*, 2010. Cited at 92

[BGW99]   L. Babai, A. Gál, and A. Wigderson. Superpolynomial lower bounds for monotone span programs. *Combinatorica*, 19(3):301–319, 1999. Cited at 85

[BH14]   S. Bravyi and M. Hastings. On complexity of the quantum Ising model. *arXiv preprint arXiv:1410.0703*, 2014. Cited at 119

[BIP16]   P. Bürgisser, C. Ikenmeyer, and G. Panova. No occurrence obstructions in geometric complex-ity theory. *arXiv preprint arXiv:1604.06431*, 2016. Cited at 131, 151

[BIW06]   B. Barak, R. Impagliazzo, and A. Wigderson. Extracting randomness using few independent sources. *SIAM Journal on computing*, 36(4):1095–1118, 2006. Cited at 102, 138

[BJ11]   L. Brouwer and Egbertus J. Über abbildung von mannigfaltigkeiten. *Mathematische Annalen*, 71(1):97–115, 1911. Cited at 227

[BJ01]   A.A. Bulatov and P. Jeavons. Algebraic structures in combinatorial problems. *International Journal of Algebra and Computing*, 2001. Cited at 41

[BJKS04]   Z. Bar-Yossef, T.S. Jayram, R. Kumar, and D Sivakumar. An information statistics approach to data stream and communication complexity. *Journal of computer and system sciences*, 68(4):702–732, 2004. Cited at 163, 172

[BJSW16]   A. Broadbent, Z. Ji, F. Song, and J. Watrous. Zero-knowledge proof systems for QMA. In *Proceedings of 57th annual IEEE Symposium on Foundations of Computer Science*, pages 31–40. IEEE, 2016. Cited at 121

[BK89]   M. Blum and S. Kannan. Designing programs that check their work. In *Proceedings of the 21st annual ACM symposium on Theory of Computing*, pages 86–97. ACM, 1989. Cited at 106

[BKI⁺96]   P. Beame, J. Krajicek, R. Impagliazzo, T. Pitassi, and P. Pudlak. Lower bounds for Hilbert's Nullstellensatz and propositional proofs. *Proceedings of the London Math Society*, 73(3):1–26, 1996. Cited at 62

[BKN14]   Z. Brakerski, Y.T. Kalai, and M. Naor. Fast interactive coding against adversarial noise. *Journal of the ACM*, 61(6):35, 2014. Cited at 177

[BKPS02]   P. Beame, R. Karp, T. Pitassi, and M. Saks. The efficiency of resolution and Davis–Putnam procedures. *SIAM Journal on computing*, 31(4):1048–1075, 2002. Cited at 68

[BKS99]   I. Benjamini, G. Kalai, and O. Schramm. Noise sensitivity of Boolean functions and applications to percolation. *Publications Mathématiques de l'Institut des Hautes Études Scientifiques*, 90(1):5–43, 1999. Cited at 145

[BKS⁺05]   B. Barak, G. Kindler, R. Shaltiel, B. Sudakov, and A. Wigderson. Simulating independence: New constructions of condensers, Ramsey graphs, dispersers, and extractors. In *Proceedings of the 37th annual ACM symposium on Theory of Computing*, pages 1–10. ACM, 2005. Cited at 84

[BKT04]   J. Bourgain, N. Katz, and T. Tao. A sum-product estimate in finite fields, and applications. *Geometric & Functional Analysis GAFA*, 14(1):27–57, 2004. Cited at 93, 138

[BKW17]   L. Barto, A. Krokhin, and R. Willard. Polymorphisms, and how to use them. In *Dagstuhl Follow-Ups*, volume 7. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. Cited at 41

[BL97]   A. L. Blum and P. Langley. Selection of relevant features and examples in machine learning. *Artificial intelligence*, 97(1-2):245–271, 1997. Cited at 200

[BL06]   Y. Bilu and N. Linial. Lifts, discrepancy and nearly optimal spectral gap. *Combinatorica*, 26(5):495–519, 2006. Cited at 93, 139

[BL08]   P. A. Brooksbank and E. M. Luks. Testing isomorphism of modules. *Journal of Algebra*, 320(11):4020–4029, 2008. Cited at 152

[BL12]   Y. Bilu and N. Linial. Are stable instances easy? *Combinatorics, Probability and Computing*, 21(05):643–660, 2012. Cited at 43, 259

[BLG12]   H. Bäärnhielm and C.R. Leedham-Green. The product replacement prospector. *Journal of Symbolic Computation*, 47(1):64–75, 2012. Cited at 142

[BLMW11]   P. Bürgisser, J.M. Landsberg, L. Manivel, and J. Weyman. An overview of mathematical issues arising in the geometric complexity theory approach to VP≠VNP. *SIAM Journal on computing*, 40(4):1179–1209, 2011. Cited at 131

[BLNPL14]   M. Babaioff, B. Lucier, N. Nisan, and R. Paes Leme. On the efficiency of the Walrasian mechanism. In *Proceedings of the 15th ACM conference on Economics and computation*, pages 783–800. ACM, 2014. Cited at 251

[BLR93]   M. Blum, M. Luby, and R. Rubinfeld. Self-testing/correcting with applications to numerical problems. *Journal of computer and system sciences*, 47(3):549–595, 1993. Cited at 106, 236

[Blu86]   M. Blum. Independent unbiased coin flips from a correlated biased source—a finite state Markov chain. *Combinatorica*, 6(2):97–108, 1986. Cited at 98

[BM84]   M. Blum and S. Micali. How to generate cryptographically secure sequences of pseudorandom bits. *SIAM Journal on Computing*, 13:850–864, 1984. Cited at 77, 207

[BO83]   M. Ben-Or. Another advantage of free choice: Completely asynchronous agreement protocols. In *Proceedings of the second annual ACM symposium on principles of distributed computing*, pages 27–30. ACM, 1983. Cited at 224

[BO85]   M. Ben-Or. Private communication, 1985. Cited at 126

[BOC92]   M. Ben-Or and R. Cleve. Computing algebraic formulas using a constant number of registers. *SIAM Journal on computing*, 21(1):54–58, 1992. Cited at 160

[BOGKW89]   M. Ben-Or, S. Goldwasser, J. Kilian, and A. Wigderson. Efficient identification schemes using two prover interactive proofs. *Advances in Cryptography (CRYPTO 89), Lecture Notes in Computing*, 435:498–506, 1989. Cited at 105, 106, 109, 122

[BOGW88]   M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the 20th annual ACM symposium on Theory of Computing*, pages 1–10. ACM, 1988. Cited at 214, 251

[BOL85]   M. Ben-Or and N. Linial. Collective coin flipping, robust voting schemes and minima of Banzhaf values. In *Proceedings of 26th annual IEEE Symposium on Foundations of Computer Science*, pages 408–416. IEEE, 1985. Cited at 145

[Bor85]   C. Borell. Geometric bounds on the Ornstein-Uhlenbeck velocity process. *Zeitschrift für Wahrscheinlichkeitstheorie und Verwandte Gebiete*, 70(1):1–13, 1985. Cited at 146

[Bou85]   J. Bourgain. On Lipschitz embedding of finite metric spaces in Hilbert space. *Israel Journal of Mathematics*, 52(1-2):46–52, 1985. Cited at 139

[BP98]   P. Beame and T. Pitassi. Propositional proof complexity: Past, present, and future. *Bulletin EATCS*, 65:66–89, 1998. Cited at 57

[BPR97]   M. Bonet, T. Pitassi, and R. Raz. Lower bounds for cutting planes proofs with small coefficients. *The Journal of Symbolic Logic*, 62(03):708–728, 1997. Cited at 68

[BPR15]   N. Bitansky, O. Paneth, and A. Rosen. On the cryptographic hardness of finding a Nash equilibrium. In *Proceedings of 56th annual IEEE Symposium on Foundations of Computer Science*, pages 1480–1498. IEEE, 2015. Cited at 250

[BR14]   M. Braverman and A. Rao. Information equals amortized communication. *IEEE Transactions on Information Theory*, 60(10):6058–6069, 2014. Cited at 173, 174, 175

[Bra84]   G. Bracha. An asynchronous $[(n-1)/3]$-resilient consensus protocol. In *Proceedings of the third annual ACM symposium on principles of distributed computing*, pages 154–162. ACM, 1984. Cited at 224

[Bra15]   M. Braverman. Interactive information complexity. *SIAM Journal on computing*, 44(6):1698–1739, 2015. Cited at 175

[Bra18]   Z. Brakerski. Fundamentals of fully homomorphic encryption — A survey. *Electronic Colloquium on Computational Complexity (ECCC)*, 25:125, 2018. Cited at 149

[Bro89]   A. Broder. Generating random spanning trees. In *Proceedings of 30th annual IEEE Symposium on Foundations of Computer Science*, pages 442–447. IEEE, 1989. Cited at 143

[Bro15]   A. Broadbent. Delegating private quantum computations. *Canadian Journal of Physics*, 93(999):1–6, 2015. Cited at 121

[BRS11]   M. Biely, P. Robinson, and U. Schmid. Easy impossibility proofs for k-set agreement in message passing systems. In *International conference On Principles Of Distributed Systems*, pages 299–312. Springer, 2011. Cited at 225

[BRSW12]   B. Barak, A. Rao, R. Shaltiel, and A. Wigderson. 2-source dispersers for $n^{o(1)}$ entropy, and Ramsey graphs beating the Frankl-Wilson construction. *Annals of Mathematics*, 176(3):1483–1543, 2012. Cited at 84

[BRW05]   R.D. Barish, P.W.K. Rothemund, and E. Winfree. Two computational primitives for algorithmic self-assembly: Copying and counting. *Nano letters*, 5(12):2586–2592, 2005. Cited at 244

[BS83]   W. Baur and V. Strassen. The complexity of partial derivatives. *Theoretical Computer Science*, 22(3):317–330, 1983. Cited at 126, 127

[BS84]   L. Babai and E. Szemerédi. On the complexity of matrix group problems I. In *Proceedings of 25th annual IEEE Symposium on Foundations of Computer Science*, pages 229–240. IEEE, 1984. Cited at 22, 141

[BS97]   E. Bach and J. Shallit. *Algorithmic Number Theory: Efficient Algorithms*, volume 1. MIT Press, Cambridge, 1997. Cited at 137

[BS14]   B. Barak and D. Steurer. Sum-of-squares proofs and the quest toward optimal algorithms. In *Proceedings of the International Congress of Mathematicians (ICM)*, 2014. Cited at 65, 235

[BSBC+17]   E. Ben-Sasson, I. Bentov, A. Chiesa, A. Gabizon, D. Genkin, M. Hamilis, E. Pergament, M. Riabzev, M. Silberstein, E. Tromer, et al. Computational integrity with a public random string from quasi-linear pcps. In *Annual International conference on the Theory and Applications of Cryptographic Techniques*, pages 551–579. Springer, 2017. Cited at 113

[BSCTV17]   E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza. Scalable zero knowledge via cycles of elliptic curves. *Algorithmica*, 79(4):1102–1160, 2017. Cited at 112

[BSS14]   J. Batson, D. A. Spielman, and N. Srivastava. Twice-Ramanujan sparsifiers. *SIAM Review*, 56(2):315–334, 2014. Cited at 138

[BSSV03]   P. Beame, M. Saks, X. Sun, and E. Vee. Time-space trade-off lower bounds for randomized computation of decision problems. *Journal of the ACM*, 50(2):154–195, 2003. Cited at 156

[BSW99]   E. Ben-Sasson and A. Wigderson. Short proofs are narrow—resolution made simple. In *Proceedings of the 31st annual ACM symposium on Theory of Computing*, pages 517–526. ACM, 1999. Cited at 67

[BSZ13]   J. Bourgain, P. Sarnak, and T. Ziegler. Disjointness of Moebius from horocycle flows. In *From Fourier analysis and number theory to Radon transforms and geometry*, pages 67–83. Springer, 2013. Cited at 85

[BT91]   J. Bourgain and L. Tzafriri. On a problem of Kadison and Singer. *Journal Fur Die Reine Und Angewandte Mathematik*, 420:1–43, 1991. Cited at 138

[Bul06]   A.A. Bulatov. A dichotomy theorem for constraint satisfaction problems on a 3-element set. *Journal of the ACM*, 53(1):66–120, 2006. Cited at 41

[Bul17]   A.A. Bulatov. A dichotomy theorem for nonuniform CSPs. *arXiv preprint arXiv:1703.03021*, 2017. Cited at 41

[Bus87]   S. Buss. Polynomial size proofs of the propositional pigeonhole principle. *Journal of Symbolic Logic*, 52:916–927, 1987. Cited at 66

[BV97]   E. Bernstein and U. Vazirani. Quantum complexity theory. *SIAM Journal on computing*, 26(5):1411–1473, 1997. Cited at 114

[BV14]   Z. Brakerski and V. Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. *SIAM Journal on computing*, 43(2):831–871, 2014. Cited at 215

[BY13]   J. Bourgain and A. Yehudayoff. Expansion in $SL_2(\mathbb{R})$ and monotone expanders. *Geometric and Functional Analysis*, 23(1):1–41, 2013. Cited at 93

[Can01]   R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings of 42nd annual IEEE Symposium on Foundations of Computer Science*, pages 136–145. IEEE, 2001. Cited at 213

[Cay45]   A. Cayley. *On the theory of linear transformations*. E. Johnson, 1845. Cited at 149

[CC12]   J. Cai and X. Chen. Complexity of counting CSP with complex weights. In *Proceedings of the 44th annual ACM symposium on Theory of Computing*, pages 909–920. ACM, 2012. Cited at 42

[CC17]   J. Cai and X. Chen. *Complexity Dichotomies for Counting Problems: Volume 1, Boolean Domain*. Cambridge University Press, 2017. Cited at 42, 237

[CCD88]   D. Chaum, C. Crépeau, and I. Damgård. Multiparty unconditionally secure protocols. In *Proceedings of the 20th annual ACM symposium on Theory of Computing*, pages 11–19. ACM, 1988. Cited at 214

[CCD+03]   A.M. Childs, R. Cleve, E. Deotto, E. Farhi, S. Gutmann, and D.A. Spielman. Exponential algorithmic speedup by a quantum walk. In *Proceedings of the 35th annual ACM symposium on Theory of Computing*, pages 59–68. ACM, 2003. Cited at 116

[CCL10]   J. Cai, X. Chen, and D. Li. Quadratic lower bound for permanent vs. determinant in any characteristic. *Computational Complexity*, 19(1):37–56, 2010. Cited at 131

[CDD+99]   R. Cramer, I. Damgård, S. Dziembowski, M. Hirt, and T. Rabin. Efficient multiparty computations secure against an adaptive adversary. In *International conference on the Theory and Applications of Cryptographic Techniques*, pages 311–326. Springer, 1999. Cited at 213

[CDNO97]   R. Canetti, C. Dwork, M. Naor, and R. Ostrovsky. Deniable encryption. In *Annual International Cryptology Conference*, pages 90–104. Springer, 1997. Cited at 207

[CDT09]   X. Chen, X. Deng, and S. Teng. Settling the complexity of computing two-player Nash equilibria. *Journal of the ACM*, 56(3):14, 2009. Cited at 38, 250

[CEI96]   M. Clegg, J. Edmonds, and R. Impagliazzo. Using the Gröbner basis algorithm to find proofs of unsatisfiability. In *Proceedings of the 28th annual ACM symposium on Theory of Computing*, pages 174–183. ACM, 1996. Cited at 62

[CG88]   B. Chor and O. Goldreich. Unbiased bits from sources of weak randomness and probabilistic communication complexity. *SIAM Journal on computing*, 17(2):230–261, 1988. Cited at 98, 102

[CGH+15]   J. Coron, C. Gentry, S. Halevi, T. Lepoint, H.K. Maji, E. Miles, M. Raykova, A. Sahai, and M. Tibouchi. Zeroizing without low-level zeroes: New MMAP attacks and their limitations. In *Annual Cryptology Conference*, pages 247–266. Springer, 2015. Cited at 216

[CGMA85]   B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults. In *Proceedings of 26th annual IEEE Symposium on Foundations of Computer Science*, pages 383–395. IEEE, 1985. Cited at 213

[CGW89]   F.R.K. Chung, R.L. Graham, and R.M. Wilson. Quasi-random graphs. *Combinatorica*, 9(4):345–362, 1989. Cited at 89

[Cha93]   S. Chaudhuri. More choices allow more faults: Set consensus problems in totally asynchronous systems. *Information and Computation*, 105(1):132–158, 1993. Cited at 225, 227

[Cha12]   B. Chazelle. Natural algorithms and influence systems. *Communications of the ACM*, 55(12):101–110, 2012. Cited at 246, 252

[Cha15]   B. Chazelle. An algorithmic approach to collective behavior. *Journal of Statistical Physics*, 158(3):514–548, 2015. Cited at 246, 252

[Che70]   J. Cheeger. A lower bound for the smallest eigenvalue of the Laplacian. *Problems in Analysis*, 625:195–199, 1970. Cited at 90

[CHHL18]   E. Chattopadhyay, P. Hatami, K. Hosseini, and S. Lovett. Pseudorandom generators from polarizing random walks. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 102. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. Cited at 79

[CHPW98]   A. Condon, L. Hellerstein, S. Pottle, and A. Wigderson. On the power of finite automata with both nondeterministic and probabilistic states. *SIAM Journal on computing*, 27(3):739–762, 1998. Cited at 160

[CHSH69]   J.F. Clauser, M.A. Horne, A. Shimony, and R.A. Holt. Proposed experiment to test local hidden-variable theories. *Physical review letters*, 23(15):880, 1969. Cited at 121, 122

[Chv73]   V. Chvátal. Edmonds polytopes and a hierarchy of combinatorial problems. *Discrete Mathemaics*, 4:305–337, 1973. Cited at 63

[CIKK16]   M.L. Carmosino, R. Impagliazzo, V. Kabanets, and A. Kolokolova. Learning algorithms from natural proofs. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 50. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. Cited at 80

[CJW06]   J.A. Carlson, A. Jaffe, and A. Wiles. *The Millennium Prize Problems*. American Mathematical Society, 2006. Cited at 24

[CKGS98]   B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan. Private information retrieval. *Journal of the ACM*, 45(6):965–981, 1998. Cited at 112, 215

[CKL13]   B.I. Carlin, S. Kogan, and R. Lowery. Trading complex assets. *The Journal of Finance*, 68(5):1937–1960, 2013. Cited at 252

[CKM+11]   P. Christiano, J.A. Kelner, A. Madry, D.A. Spielman, and S. Teng. Electrical flows, Laplacian systems, and faster approximation of maximum flow in undirected graphs. In *Proceedings of the 43rd annual ACM symposium on Theory of Computing*, pages 273–282. ACM, 2011. Cited at 235

[CKSU05]   H. Cohn, R. Kleinberg, B. Szegedy, and C. Umans. Group-theoretic algorithms for matrix multiplication. In *Proceedings of 46th annual IEEE Symposium on Foundations of Computer Science*, pages 379–388. IEEE, 2005. Cited at 128

[CKW11]   X. Chen, N. Kayal, and A. Wigderson. *Partial derivatives in arithmetic complexity and beyond*. Now Publishers Inc., 2011. Cited at 124, 134

[CL16]   E. Chattopadhyay and X. Li. Extractors for sumset sources. In *Proceedings of the 48th annual ACM symposium on Theory of Computing*, pages 299–311. ACM, 2016. Cited at 102

[CLO92]   D. Cox, J. Little, and D. O'Shea. *Ideals, Varieties, and Algorithms: an introduction to computational algebraic geometry and commutative algebra*. Undergraduate Texts in Mathematics. Springer-Verlag, New York, 1992. Cited at 149

[CLPV13]   E. Chastain, A. Livnat, C. Papadimitriou, and U. Vazirani. Multiplicative updates in coordination games and the theory of evolution. In *Proceedings of the 4th conference on innovations in theoretical Computer Science*, pages 57–58. ACM, 2013. Cited at 182

[CLPV14]   E. Chastain, A. Livnat, C. Papadimitriou, and U. Vazirani. Algorithms, games, and evolution. *Proceedings of the National Academy of Sciences*, 111(29):10620–10623, 2014. Cited at 36, 246

[CLR01]   T.H. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, MA, McGraw-Hill Book Co., New York, 2001. Cited at 18

[CLRS16]   S.O Chan, J.R. Lee, P. Raghavendra, and D. Steurer. Approximate constraint satisfaction requires large LP relaxations. *Journal of the ACM*, 63(4):34, 2016. Cited at 235

[CM84]   K.M. Chandy and J. Misra. The drinking philosophers problem. *ACM Transactions on Programming Languages and Systems*, 6(4):632–646, 1984. Cited at 221

[CM13]   T. Cubitt and A. Montanaro. Complexity classification of local Hamiltonian problems. *arXiv preprint arXiv:1311.3161*, 2013. Cited at 119

[Cob65]   A. Cobham. The intrinsic computational difficulty of functions. *Logic, Methodology, and Philosophy of Science: Proceedings of the 1964 International Congress*, pages 24–30, 1965. Cited at 17

[Coh16]   M.B. Cohen. Ramanujan graphs in polynomial time. *arXiv preprint arXiv:1604.03544*, 2016. Cited at 93

[Coh17]   G. Cohen. Towards optimal two-source extractors and Ramsey graphs. In *Proceedings of the 49th annual ACM symposium on Theory of Computing*, pages 1157–1170. ACM, 2017. Cited at 84

[Col06]   Roger A. Colbeck. *Quantum and relativistic protocols for secure multi-party computation*. PhD thesis, Trinity College, University of Cambridge, 2006. Cited at 123

[Con92]   A. Condon. The complexity of stochastic games. *Information and Computation*, 96(2):203–224, 1992. Cited at 28

[Con93]   A. Condon. On algorithms for simple stochastic games. *Advances in computational complexity theory*, 13:51–73, 1993. Cited at 40

[Coo71]   S.A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd annual ACM symposium on Theory of Computing*, pages 151–158. ACM, 1971. Cited at 22, 30, 31

[Coo02]   G. Cooperman. Towards a practical, theoretically sound algorithm for random generation in finite groups. *arXiv preprint arXiv:0205203*, 2002. Cited at 142

[Cor11]   Graham Cormode. Sketch techniques for approximate query processing. *Foundations and Trends in Databases*, 2011. Cited at 156

[Cov69]   R.R. Coveyou. Random number generation is too important to be left to chance. *Studies in applied mathematics*, 3:70–111, 1969. Cited at 77

[Cov91]   T.M. Cover. Universal portfolios. *Mathematical finance*, 1(1):1–29, 1991. Cited at 184

[CR79]   S.A. Cook and R.A. Reckhow. The relative efficiency of propositional proof systems. *Journal of Symbolic Logic*, 44:36–50, 1979. Cited at 60, 61, 66

[CRVW02]   M. Capalbo, O. Reingold, S. Vadhan, and A. Wigderson. Randomness conductors and constant-degree lossless expanders. In *Proceedings of the 34th annual ACM symposium on Theory of Computing*, volume 155, pages 659–668. ACM, 2002. Cited at 92, 236

[CS88]   V. Chvátal and E. Szemerédi. Many hard examples for Resolution. *Journal of the ACM*, 35(4):759–768, 1988. Cited at 67

[CS07]   A. Czumaj and C. Sohler. Sublinear-time approximation algorithms for clustering via random sampling. *Random Structures & Algorithms*, 30(1-2):226–256, 2007. Cited at 112

[CSWY01]   A. Chakrabarti, Y. Shi, A. Wirth, and A. Yao. Informational complexity and the direct sum problem for simultaneous message complexity. In *Proceedings of 42nd annual IEEE Symposium on Foundations of Computer Science*, pages 270–278. IEEE, 2001. Cited at 172, 173

[CU03]   H. Cohn and C. Umans. A group-theoretic approach to fast matrix multiplication. In *Proceedings of 44th annual IEEE Symposium on Foundations of Computer Science*, pages 438–449. IEEE, 2003. Cited at 128

[CU13]   H. Cohn and C. Umans. Fast matrix multiplication using coherent configurations. In *Proceedings of the 24th annual ACM-SIAM symposium on discrete algorithms*, pages 1074–1086. SIAM, 2013. Cited at 128

[CV86]   R. Cole and U. Vishkin. Deterministic coin tossing with applications to optimal parallel list ranking. *Information and Control*, 70(1):32–53, 1986. Cited at 231

[CvD10]   A.M. Childs and W. van Dam. Quantum algorithms for algebraic problems. *Reviews of Modern Physics*, 82(1):1, 2010. Cited at 116

[CVZ18]   M. Christandl, P. Vrana, and J. Zuiddam. Barriers for fast matrix multiplication from irreversibility. *arXiv preprint arXiv:1812.06952*, 2018. Cited at 128

[CW89]   A. Cohen and A. Wigderson. Dispersers, deterministic amplification, and weak random sources. In *Proceedings of 30th annual IEEE Symposium on Foundations of Computer Science*, pages 14–19. IEEE, 1989. Cited at 101

[CY14]   M. Coudron and H. Yuen. Infinite randomness expansion with a constant number of devices. In *Proceedings of the 46th annual ACM symposium on Theory of Computing*, pages 427–436. ACM, 2014. Cited at 123

[CZ12]   J.I. Cirac and P. Zoller. Goals and opportunities in quantum simulation. *Nature Physics*, 8(4):264–266, 2012. Cited at 119

[CZ16]   E. Chattopadhyay and D. Zuckerman. Explicit two-source extractors and resilient functions. In *Proceedings of the 48th annual ACM symposium on Theory of Computing*, pages 670–683. ACM, 2016. Cited at 84, 102

[DA01]   P. Dayan and L. F. Abbott. *Theoretical neuroscience*, volume 806. MIT Press, 2001. Cited at 247

[Dav71]   R.O. Davies. Some remarks on the Kakeya problem. In *Mathematical proceedings of the Cambridge Philosophical Society*, volume 69, pages 417–421. Cambridge University Press, 1971. Cited at 137

[DDMO04] E.D. Demaine, S.L. Devadoss, J.S.B. Mitchell, and J. O'Rourke. Continuous foldability of polygonal paper. In *CCCG*, pages 64–67, 2004. Cited at 245

[DDN03]   D. Dolev, C. Dwork, and M. Naor. Nonmalleable cryptography. *SIAM review*, 45(4):727–784, 2003. Cited at 207

[DdW09]   A. Drucker and R. de Wolf. Quantum proofs for classical theorems. *arXiv preprint arXiv:0910.3376*, 2009. Cited at 122

[Del74]   P. Deligne. La conjecture de Weil. I. *Publications Mathématiques de l'Institut des Hautes Études Scientifiques*, 43(1):273–307, 1974. Cited at 85

[Del80]   P. Deligne. La conjecture de Weil. II. *Publications mathématiques de l'IHÉS*, 52(1):137–252, 1980. Cited at 85

[Der17]   Z. Derakhshandeh. *Algorithmic Foundations of Self-Organizing Programmable Matter*. PhD thesis, Arizona State University, 2017. Cited at 244

[Deu85]   D. Deutsch. Quantum theory, the Church-Turing principle and the universal quantum computer. In *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, volume 400, pages 97–117. The Royal Society, 1985. Cited at 114, 242

[DFH+15] C. Dwork, V. Feldman, M. Hardt, T. Pitassi, O. Reingold, and A.L. Roth. Preserving statistical validity in adaptive data analysis. In *Proceedings of the 47th annual ACM symposium on Theory of Computing*, pages 117–126. ACM, 2015. Cited at 254

[DFK91]   M. Dyer, A. Frieze, and R. Kannan. A random polynomial-time algorithm for approximating the volume of convex bodies. *Journal of the ACM*, 38(1):1–17, 1991. Cited at 72, 143, 144

[DFP04]   M. Duflot, L. Fribourg, and C. Picaronny. Randomized dining philosophers without fairness assumption. *Distributed Computing*, 17(1):65–76, 2004. Cited at 222

[DG16]   Z. Dvir and S. Gopi. 2-server PIR with subpolynomial communication. *Journal of the ACM*, 63(4):39, 2016. Cited at 112, 236

[DGP09]   C. Daskalakis, P.W. Goldberg, and C.H. Papadimitriou. The complexity of computing a Nash equilibrium. *SIAM Journal on computing*, 39(1):195–259, 2009. Cited at 38, 250

[DGW09]   Z. Dvir, A. Gabizon, and A. Wigderson. Extractors and rank extractors for polynomial sources. *Computational Complexity*, 18(1):1–58, 2009. Cited at 102

[DH76]   W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions Information Theory*, 22:644–654, 1976. Cited at 45, 46, 203, 206

[DHK+19] I. Dinur, P. Harsha, T. Kaufman, I. Livni, and A. Ta-Shma. List decoding with double samplers. In *Proceedings of the Thirtieth annual ACM-SIAM Symposium on Discrete Algorithms, 2019*, pages 2134–2153, 2019. Cited at 93

[Dia88]   P. Diaconis. Group representations in probability and statistics. *Lecture Notes-Monograph Series*, 11:i–192, 1988. Cited at 142

[Dij65]   E.W. Dijkstra. Solution of a problem in concurrent programming control. *Communications of the ACM*, 8(9):569, 1965. Cited at 220

[Dij71]   E.W. Dijkstra. Hierarchical ordering of sequential processes. *Acta informatica*, 1(2):115–138, 1971. Cited at 220

[Din07]　　I. Dinur. The PCP Theorem by gap amplification. *Journal of the ACM*, 54(12), 2007. Cited at 92, 110

[Dix08]　　J.D. Dixon. Generating random elements in finite groups. *The electronic journal of combinatorics*, 13(R94):1, 2008. Cited at 142

[DK15]　　H. Derksen and G. Kemper. *Computational invariant theory*. Springer, 2015. Cited at 149

[DK16]　　V. Danos and H. Koeppl. Self-assembly and self-organization in computer science and biology (Dagstuhl Seminar 15402). In *Dagstuhl Reports*, volume 5, pages 125–138. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. Cited at 244

[DK17]　　I. Dinur and T. Kaufman. High dimensional expanders imply agreement expanders. *Electronic Colloquium on Computational Complexity (ECCC)*, 24:89, 2017. Cited at 93, 236

[DKSS13]　Z. Dvir, S. Kopparty, S. Saraf, and M. Sudan. Extensions to the method of multiplicities, with applications to Kakeya sets and mergers. *SIAM Journal on computing*, 42(6):2305–2328, 2013. Cited at 137

[DL78]　　R.A. DeMillo and R.J. Lipton. A probabilistic remark on algebraic program testing. *Information Processing Letters*, 7(4):193–195, 1978. Cited at 70

[DLL62]　　M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Journal of the ACM*, 5(7):394–397, 1962. Cited at 67

[DM15]　　H. Derksen and V. Makam. Polynomial degree bounds for matrix semi-invariants. *arXiv preprint arXiv:1512.03393*, 2015. Cited at 151, 152

[DM16]　　I. Dinur and O. Meir. Toward the KRW composition conjecture: cubic formula lower bounds via communication complexity. In *Proceedings of the 31st conference on computational complexity*, page 3. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. Cited at 53

[DO08]　　E.D. Demaine and J. O'Rourke. *Geometric folding algorithms: linkages, origami, polyhedra*. Cambridge University Press, 2008. Cited at 245

[Don92]　　S. Donkin. Invariants of several matrices. *Inventiones mathematicae*, 110(1):389–401, 1992. Cited at 151

[DR14]　　C. Dwork and A. Roth. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3-4), 2014. Cited at 202

[DR16]　　D. Dadush and O. Regev. Towards strong reverse Minkowski-type inequalities for lattices. In *Proceedings of 57th annual IEEE Symposium on Foundations of Computer Science*, pages 447–456. IEEE, 2016. Cited at 147

[DS90]　　C. Dwork and L. Stockmeyer. A time complexity gap for two-way probabilistic finite-state automata. *SIAM Journal on computing*, 19(6):1011–1023, 1990. Cited at 160

[DS92]　　C. Dwork and L. Stockmeyer. Finite state verifiers I: the power of interaction. *Journal of the ACM*, 39(4):800–828, 1992. Cited at 160

[DS07]　　Z. Dvir and A. Shpilka. Locally decodable codes with two queries and polynomial identity testing for depth 3 circuits. *SIAM Journal on computing*, 36(5):1404–1434, 2007. Cited at 138

[DS13]　　A. De and R. Servedio. Efficient deterministic approximate counting for low-degree polynomial threshold functions. *arXiv preprint arXiv:1311.7178*, 2013. Cited at 95

[DSS14] A. Daniely and S. Shalev-Shwartz. Complexity theoretic limitations on learning DNF's. *CoRR, abs/1404.3378*, 1(2.1):2–1, 2014. Cited at 200

[DSTW14] I. Diakonikolas, R.A. Servedio, L. Tan, and A. Wan. A regularity lemma and low-weight approximators for low-degree polynomial threshold functions. *Theory of Computing*, 10(2):27–53, 2014. Cited at 96

[DSVW04] M. Dyer, A. Sinclair, E. Vigoda, and D. Weitz. Mixing in time and space for lattice spin systems: A combinatorial view. *Random Structures & Algorithms*, 24(4):461–479, 2004. Cited at 144, 238

[DSW14a] Z. Dvir, S. Saraf, and A. Wigderson. Breaking the quadratic barrier for 3-LCC's over the reals. In *Proceedings of the 46th annual ACM symposium on Theory of Computing*, pages 784–793. ACM, 2014. Cited at 112

[DSW14b] Z. Dvir, S. Saraf, and A. Wigderson. Improved rank bounds for design matrices and a new proof of Kelly's theorem. In *Forum of Mathematics, Sigma*, volume 2. Cambridge University Press, 2014. Cited at 112, 236

[Dvi09] Z. Dvir. On the size of Kakeya sets in finite fields. *Journal of the American Mathematical Society*, 22(4):1093–1097, 2009. Cited at 101, 137

[Dvi10] Z. Dvir. Guest column: from randomness extraction to rotating needles. *ACM SIGACT News*, 40(4):46–61, 2010. Cited at 137

[DW00] H. Derksen and J. Weyman. Semi-invariants of quivers and saturation for Littlewood-Richardson coefficients. *Journal of the American Mathematical Society*, 13(3):467–479, 2000. Cited at 152

[DW06] H. Derksen and J. Weyman. The combinatorics of quiver representations. *arXiv preprint math/0608288*, 2006. Cited at 151

[DW11] Z. Dvir and A. Wigderson. Kakeya sets, new mergers, and old extractors. *SIAM Journal on computing*, 40(3):778–792, 2011. Cited at 101

[DZ01] M. Domokos and A.N. Zubkov. Semi-invariants of quivers as determinants. *Transformation groups*, 6(1):9–24, 2001. Cited at 152

[Edm65a] J. Edmonds. Maximum matching and a polyhedron with 0, l-vertices. *Jornal of Research of the National Bureau of Standards B*, 69(1965):125–130, 1965. Cited at 169

[Edm65b] J. Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17(3):449–467, 1965. Cited at 17, 19, 169

[Edm66] J. Edmonds. Minimum partition of a matroid into independent sets. *Journal of Research of the National Bureau of Standards*, B(69):67–72, 1966. Cited at 17, 23, 26, 27

[Edm67a] J. Edmonds. Optimum branchings. *Journal of Research of the National Bureau of Standards B*, 71:233, 1967. Cited at 17, 23

[Edm67b] J. Edmonds. Systems of distinct representatives and linear algebra. *J. Res. Nat. Bur. Standards Sect. B*, 71:241–245, 1967. Cited at 132

[Efr12] K. Efremenko. 3-query locally decodable codes of subexponential length. *SIAM Journal on computing*, 41(6):1694–1703, 2012. Cited at 236

[EGL85] S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. *Communications of the ACM*, 28(6):637–647, 1985. Cited at 213

[EGO84]   A. El Gamal and A. Orlitsky. Interactive data compression. In *Proceedings of 25th annual IEEE Symposium on Foundations of Computer Science*, pages 100–108. IEEE, 1984. Cited at 174

[EGOW17]   K. Efremenko, A. Garg, R. Oliveira, and A. Wigderson. Barriers for rank methods in arithmetic complexity. *arXiv preprint arXiv:1710.09502*, 2017. Cited at 132, 133

[EGSZ16]   F. Ellen, R. Gelashvili, N. Shavit, and L. Zhu. A complexity-based hierarchy for multiprocessor synchronization. *arXiv preprint arXiv:1607.06139*, 2016. Cited at 219

[EHKS14]   K. Eisenträger, S. Hallgren, A. Kitaev, and F. Song. A quantum algorithm for computing the unit group of an arbitrary degree number field. In *Proceedings of the 46th annual ACM symposium on Theory of Computing*, pages 293–302. ACM, 2014. Cited at 116

[EK10]   S. Easley and J. Kleinberg. *Networks, crowds, and markets: Reasoning about a highly connected world*. Cambridge University Press, 2010. Cited at 36, 252

[EK16]   S. Evra and T. Kaufman. Bounded degree cosystolic expanders of every dimension. In *Proceedings of the 48th annual ACM Symposium on Theory of Computing, 2016*, pages 36–48, 2016. Cited at 93

[Eke91]   A.K. Ekert. Quantum cryptography based on Bell's theorem. *Physical review letters*, 67(6):661, 1991. Cited at 118

[Elg85]   T. Elgamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985. Cited at 46

[Eli57]   P. Elias. List decoding for noisy channels. *Research Laboratory of Electronics, Massachusetts Institute of Technology*, 1957. Cited at 235

[EPR35]   A. Einstein, B. Podolsky, and N. Rosen. Can quantum-mechanical description of physical reality be considered complete? *Physical review*, 47(10):777, 1935. Cited at 121

[ER59]   P. Erdős and A. Rényi. On random graphs, I. *Publicationes Mathematicae*, 6:290–297, 1959. Cited at 89

[ER60]   P. Erdős and A. Rényi. On the evolution of random graphs. *Publications of the Mathematical Institute of the Hungarian Academy of Sciences*, 5:17–61, 1960. Cited at 89

[Erd47]   P. Erdős. Some remarks on the theory of graphs. *Bulletin of the American Mathematical Society*, 53(4):292–294, 1947. Cited at 82

[Fár48]   I. Fáry. On straight-line representation of plane graphs. *Acta Scientarum Mathematicarum (Szeged)*, 11:229–233, 1948. Cited at 16

[Fel71]   W. Feller. An introduction to probability theory and its applications. *Wiley series in probability and mathematical statistics*, 1971. Cited at 145

[Fel93]   M. R. Fellows. Computer science and mathematics in the elementary schools. In Naomi D. Fisher, Harvey B. Keynes, and Philip D. Wagreich, editors, *Proceedings of the Mathematicians and Education Reform Workshop, Seattle, 1991*, volume 3 of *Issues in Mathematics Education*, pages 143–163. Conference Board of the Mathematical Sciences, 1993. https://larc.unt.edu/ian/research/cseducation/fellows1991.pdf. Cited at 263

[Fey82]   R.P. Feynman. Simulating physics with computers. *International journal of theoretical physics*, 21(6):467–488, 1982. Cited at 114, 242, 243

[Fey86]   R.P. Feynman. Quantum mechanical computers. *Foundations of physics*, 16(6):507–531, 1986. Cited at 243

[FFN14]   B. Fisch, D. Freund, and M. Naor. Physical zero-knowledge proofs of physical properties. In *International Cryptology Conference*, pages 313–336. Springer, 2014. Cited at 113

[FG98]   E. Farhi and S. Gutmann. Quantum computation and decision trees. *Physical Review A*, 58(2):915, 1998. Cited at 116

[FGGS00] E. Farhi, J. Goldstone, S. Gutmann, and M. Sipser. Quantum computation by adiabatic evolution. *arXiv preprint quant-ph/0001106*, 2000. Cited at 120

[FGL⁺96] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy. Interactive proofs and the hardness of approximating cliques. *Journal of the ACM*, pages 268–292, 1996. Cited at 110, 235

[FHL80]   M. Furst, J. Hopcroft, and E. Luks. Polynomial-time algorithms for permutation groups. In *Proceedings of 21st annual IEEE Symposium on Foundations of Computer Science*, pages 36–41. IEEE, 1980. Cited at 20

[FK96]   A. Frieze and R. Kannan. The regularity lemma and approximation schemes for dense problems. In *Proceedings of 37th annual IEEE Symposium on Foundations of Computer Science*, pages 12–20. IEEE, 1996. Cited at 95

[FK01]   U. Feige and J. Kilian. Heuristics for semirandom graph problems. *Journal of computer and system sciences*, 63(4):639–671, 2001. Cited at 43, 259

[FKL⁺91] A. Fiat, R. M. Karp, M. Luby, L. A McGeoch, D. D. Sleator, and N. E. Young. Competitive paging algorithms. *Journal of Algorithms*, 12(4):685–699, 1991. Cited at 181

[FKLW03] M. Freedman, A. Kitaev, M. Larsen, and Z. Wang. Topological quantum computation. *Bulletin of the American Mathematical Society*, 40(1):31–38, 2003. Cited at 120

[FKO07]   U. Feige, G. Kindler, and R. O'Donnell. Understanding parallel repetition requires understanding foams. In *Proceedings of the 22nd annual IEEE conference on computational complexity*, pages 179–192. IEEE, 2007. Cited at 146

[FLP85]   M.J. Fischer, N.A. Lynch, and M.S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, 1985. Cited at 223, 225

[FLvMV05] L. Fortnow, R. Lipton, D. van Melkebeek, and A. Viglas. Time-space lower bounds for satisfiability. *Journal of the ACM*, 52(6):835–865, 2005. Cited at 156

[FMP⁺15] S. Fiorini, S. Massar, S. Pokutta, H.R. Tiwary, and R. Wolf. Exponential lower bounds for polytopes in combinatorial optimization. *Journal of the ACM*, 62(2):17, 2015. Cited at 170, 235

[For66]   G.D. Forney. *Concatenated codes*, volume 11. Citeseer, 1966. Cited at 84, 236

[For84]   E. Formanek. Invariants and the ring of generic matrices. *Journal of Algebra*, 89(1):178–223, 1984. Cited at 151

[For94]   L. Fortnow. The role of relativization in complexity theory. *Bulletin EATCS*, 52:229–244, 1994. Cited at 48

[For00]   L. Fortnow. Time–space tradeoffs for satisfiability. *Journal of computer and system sciences*, 60(2):337–353, 2000. Cited at 48, 156

[For13]   L. Fortnow. *The golden ticket: P, NP, and the search for the impossible*. Princeton University Press, 2013. Cited at 25

[FR03]   F. Fich and E. Ruppert. Hundreds of impossibility results for distributed computing. *Distributed computing*, 16(2-3):121–163, 2003. Cited at 218

[Fre81]   R. Freivalds. Probabilistic two-way machines. In *International Symposium on Mathematical Foundations of Computer Science*, pages 33–45. Springer, 1981. Cited at 159, 160

[Fre90]   Y. Freund. Boosting a weak learning algorithm by majority. In *Proceedings of the 3rd Annual Workshop on Computational Learning Theory*, volume 90, pages 202–216, 1990. Cited at 197

[FRS88]   L. Fortnow, J. Rompel, and M. Sipser. On the power of multi-power interactive protocols. In *Proceedings of Third annual Structure in Complexity Theory Conference, 1988.*, pages 156–161. IEEE, 1988. Cited at 109

[FS95]   Y. Freund and R.E. Schapire. A desicion-theoretic generalization of on-line learning and an application to boosting. In *European conference on computational learning theory*, pages 23–37. Springer, 1995. Cited at 198

[FS99]   Y. Freund and R.E. Schapire. Adaptive game playing using multiplicative weights. *Games and Economic Behavior*, 29(1):79–103, 1999. Cited at 183, 184

[FS13]   M.A. Forbes and A. Shpilka. Explicit Noether normalization for simultaneous conjugation via polynomial identity testing. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 527–542. Springer, 2013. Cited at 151, 152

[FSS84]   M. Furst, J. Saxe, and M. Sipser. Parity, circuits and the polynomial time hierarchy. *Math. Systems Theory*, 17:13–27, 1984. Cited at 52, 55

[FSV17]   M.A. Forbes, A. Shpilka, and B.L. Volk. Succinct hitting sets and barriers to proving algebraic circuits lower bounds. *arXiv preprint arXiv:1701.05328*, 2017. Cited at 132

[Für09]   M. Fürer. Faster integer multiplication. *SIAM Journal on computing*, 39(3):979–1005, 2009. Cited at 51

[FV98]   T. Feder and M.Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: A study through Datalog and group theory. *SIAM Journal on computing*, 28(1):57–104, 1998. Cited at 41

[FW81]   P. Frankl and R.M. Wilson. Intersection theorems with geometric consequences. *Combinatorica*, 1(4):357–368, 1981. Cited at 84

[Gab72]   P. Gabriel. Unzerlegbare darstellungen I. *Manuscripta Mathematica*, 6(1):71–103, 1972. Cited at 151

[Gal62]   R. Gallager. Low-density parity-check codes. *IRE Transactions on information theory*, 8(1):21–28, 1962. Cited at 236

[GBC16]   I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT Press, 2016. Cited at 185, 256

[GBG14]   A. Glaser, B. Barak, and R.J. Goldston. A zero-knowledge protocol for nuclear warhead verification. *Nature*, 510(7506):497, 2014. Cited at 113

[GBGL10]   T. Gowers, J. Barrow-Green, and I. Leader. *The Princeton companion to mathematics*. Princeton University Press, 2010. Cited at xiii

[Gel15]   R. Gelles. Coding for interactive communication: a survey. *Survey*, 2015. `http://www.cs.princeton.edu/~rgelles/papers/survey.pdf`. Cited at 176

[Gen09a]   C. Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. Cited at 214

[Gen09b]   C. Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st annual ACM symposium on Theory of Computing*, volume 9, pages 169–178, 2009. Cited at 149, 214

[GG16]   O. Goldreich and T. Gur. Universal locally testable codes. In *Electronic Colloquium on Computational Complexity (ECCC)*, volume 23, page 1, 2016. Cited at 112

[GGH⁺13]   S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B. Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *Proceedings of 54th annual IEEE Symposium on Foundations of Computer Science*, pages 40–49. IEEE, 2013. Cited at 216

[GGHR14]   S. Garg, C. Gentry, S. Halevi, and M. Raykova. Two-round secure MPC from indistinguishability obfuscation. In *Theory of Cryptography Conference*, pages 74–94. Springer, 2014. Cited at 216

[GGM86]   O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *Journal of the ACM*, 33(4):792–807, 1986. Cited at 78, 86, 200

[GGOW15]   A. Garg, L. Gurvits, R. Oliveira, and A. Wigderson. A deterministic polynomial time algorithm for non-commutative rational identity testing. *arXiv preprint arXiv:1511.03730*, 2015. Cited at 132, 153, 235

[GGOW16]   A. Garg, L. Gurvits, R. Oliveira, and A. Wigderson. Algorithmic aspects of Brascamp-Lieb inequalities. *arXiv preprint arXiv:1607.06711*, 2016. Cited at 153

[GGR98]   O. Goldreich, S. Goldwasser, and D. Ron. Property testing and its connection to learning and approximation. *Journal of the ACM (JACM)*, 45(4):653–750, 1998. Cited at 112

[GH14]   M. Ghaffari and B. Haeupler. Optimal error rates for interactive coding II: Efficiency and list decoding. In *Proceedings of 55th annual IEEE Symposium on Foundations of Computer Science*, pages 394–403. IEEE, 2014. Cited at 177

[GHK⁺16]   R. Gelles, B. Haeupler, G. Kol, N. Ron-Zewi, and A. Wigderson. Towards optimal deterministic coding for interactive communication. In *Proceedings of the 27th annual ACM-SIAM symposium on discrete algorithms*, pages 1922–1936. SIAM, 2016. Cited at 177

[GHL14]   S. Gharibian, Y. Huang, and Z. Landau. Quantum Hamiltonian complexity. *arXiv preprint arXiv:1401.3916*, 2014. Cited at 118

[GHSY12]   P. Gopalan, C. Huang, H. Simitci, and S. Yekhanin. On the locality of codeword symbols. *IEEE Transactions on Information Theory*, 58(11):6925–6934, 2012. Cited at 236

[Gil52]   E.N. Gilbert. A comparison of signalling alphabets. *Bell System Technical Journal*, 31(3):504–522, 1952. Cited at 176

[Gil77]   J. Gill. Computational complexity of probabilistic Turing machines. *SIAM Journal on Computing*, 6:675–695, 1977. Cited at 71

[Gil98]   D. Gillman. A Chernoff bound for random walks on expander graphs. *SIAM Journal on computing*, 27(4):1203–1220, 1998. Cited at 101

[GJ79]   M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York, 1979. Cited at 31, 33

[GJL16]   H. Guo, M. Jerrum, and J. Liu. Uniform sampling through the Lovász local lemma. *arXiv preprint arXiv:1611.01647*, 2016. Cited at 144, 238

[GK86]   S. Goldwasser and J. Kilian. Almost all primes can be quickly certified. In *Proceedings of the 18th annual ACM symposium on Theory of Computing*, pages 316–329. ACM, 1986. Cited at 136

[GK10]   L. Guth and N.H. Katz. On the Erdős distinct distance problem in the plane. *arXiv preprint arXiv:1011.4105*, 2010. Cited at 137

[GK16]   S. Goldwasser and Y.T. Kalai. Cryptographic assumptions: A position paper. In *Theory of Cryptography Conference*, pages 505–522. Springer, 2016. Cited at 204, 260

[GKKS13]   A. Gupta, P. Kamath, N. Kayal, and R. Saptharishi. Approaching the chasm at depth four. In *Proceedings of the 28th IEEE conference on Computational complexity*, pages 65–73. IEEE, 2013. Cited at 133, 134

[GKM+00]   Y. Gertner, S. Kannan, T. Malkin, O. Reingold, and M. Viswanathan. The relationship between public key encryption and oblivious transfer. In *Proceedings of 41st annual IEEE Symposium on Foundations of Computer Science*, pages 325–335. IEEE, 2000. Cited at 211

[GKR08]   S. Goldwasser, Y.T. Kalai, and G.N. Rothblum. Delegating computation: interactive proofs for muggles. In *Proceedings of the 40th annual ACM symposium on Theory of Computing*, pages 113–122. ACM, 2008. Cited at 215

[GKR14]   A. Ganor, G. Kol, and R. Raz. Exponential separation of information and communication. In *Proceedings of 55th annual IEEE Symposium on Foundations of Computer Science*, pages 176–185. IEEE, 2014. Cited at 175

[GKR15]   A. Ganor, G. Kol, and R. Raz. Exponential separation of information and communication. In *Symposium on the Theory of Computing*, pages 557–?566. ACM, 2015. Cited at 175

[GKSS17]   J. A. Grochow, M. Kumar, M. Saks, and S. Saraf. Towards an algebraic natural proofs barrier via polynomial identity testing. *arXiv preprint arXiv:1701.01717*, 2017. Cited at 132

[GL89]   O. Goldreich and L.A. Levin. A hard-core predicate for all one-way functions. In *Proceedings of the 21st annual ACM symposium on Theory of Computing*, pages 25–32. ACM, 1989. Cited at 236

[GLR10]   V. Guruswami, J.R. Lee, and A. Razborov. Almost Euclidean subspaces of ℓ_1^n via expander codes. *Combinatorica*, 30(1):47–68, 2010. Cited at 138

[GM82]   S. Goldwasser and S. Micali. Probabilistic encryption & how to play mental poker keeping secret all partial information. In *Proceedings of the 14th annual ACM symposium on Theory of Computing*, pages 365–377. ACM, 1982. Cited at 210

[GM84]   S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of computer and system sciences*, 28:270–299, 1984. Cited at 46, 73, 75, 76, 206, 207, 208

[GMR89]   S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989. Cited at 104, 107, 141

[GMR+12]   P. Gopalan, R. Meka, O. Reingold, L. Trevisan, and S. Vadhan. Better pseudorandom generators from milder pseudorandom restrictions. In *Proceedings of 53rd annual IEEE Symposium on Foundations of Computer Science*, pages 120–129. IEEE, 2012. Cited at 79

[GMW87]   O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proceedings of the 19th annual ACM symposium on Theory of Computing*, pages 218–229. ACM, 1987. Cited at 46, 108, 160, 210, 211, 213, 251

[GMW91]   O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity, or all languages in NP have zero-knowledge proof systems. *Journal of the ACM*, 38(1):691–729, 1991. Cited at 103, 107, 108, 209, 212, 213, 254

[GMWW14] D. Gavinsky, O. Meir, O. Weinstein, and A. Wigderson. Toward better formula lower bounds: an information complexity approach to the KRW composition conjecture. In *Proceedings of the 46th annual ACM symposium on Theory of Computing*, pages 213–222. ACM, 2014. Cited at 53

[GNW11]   O. Goldreich, N. Nisan, and A. Wigderson. On Yao's XOR-lemma. In *Studies in complexity and cryptography. Miscellanea on the interplay between randomness and computation*, pages 273–301. Springer, 2011. Cited at 80

[Goe97]   M.X. Goemans. Semidefinite programming in combinatorial optimization. *Mathematical Programming*, 79(1-3):143–161, 1997. Cited at 139

[Gol67]   E.M. Gold. Language identification in the limit. *Information and control*, 10(5):447–474, 1967. Cited at 190

[Gol97]   O. Goldreich. Notes on Levin's theory of average-case complexity. *ECCC*, TR97-058, 1997. Cited at 44

[Gol99]   O. Goldreich. Modern cryptography, probabilistic proofs and pseudorandomness. *Algorithms and Combinatorics*, 17, 1999. Cited at 70, 73, 77, 103

[Gol04]   O. Goldreich. *Foundations of Cryptography*. Cambridge University Press, Cambridge, 2001; 2004. I. Basic Tools; II. Basic Applications. Cited at 44, 75, 77, 202, 205, 212

[Gol08]   O. Goldreich. *Computational complexity: a conceptual perspective*. Cambridge University Press, Cambridge, 2008. Cited at 5, 73

[Gol10]   O. Goldreich. *Property testing: current research and surveys*, volume 6390. Springer, 2010. Cited at 90

[Gol11]   O. Goldreich. Short locally testable codes and proofs. In *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation*, pages 333–372. Springer, 2011. Cited at 112

[Gol17]   O. Goldreich. *Introduction to property testing*. Cambridge University Press, 2017. Cited at 112

[Gow01]   W.T. Gowers. A new proof of Szemerédi's theorem. *Geometric and Functional Analysis*, 11(3):465–588, 2001. Cited at 94

[GPW15]   M. Göös, T. Pitassi, and T. Watson. The landscape of communication complexity classes. *Computational Complexity*, pages 1–60, 2015. Cited at 164

[GPW17]   M. Göös, T. Pitassi, and T. Watson. Query-to-communication lifting for bpp. In *Proceedings of 58th annual IEEE Symposium on Foundations of Computer Science*, pages 132–143. IEEE, 2017. Cited at 164

[GR07]   S. Goldwasser and G.N. Rothblum. On best-possible obfuscation. In *Theory of Cryptography Conference*, pages 194–213. Springer, 2007. Cited at 216

[GR08a]   A. Gabizon and R. Raz. Deterministic extractors for affine sources over large fields. *Combinatorica*, 28(4):415–440, 2008. Cited at 102

[GR08b]   V. Guruswami and A. Rudra. Explicit codes achieving list decoding capacity: Error-correction with optimal redundancy. *IEEE Transactions on Information Theory*, 54(1):135–150, 2008. Cited at 101, 235

[GR18]   O. Goldreich and G.N. Rothblum. Simple doubly-efficient interactive proof systems for locally-characterizable sets. In *LIPIcs-Leibniz international proceedings in informatics*, volume 94. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. Cited at 215

[Gra05]   A. Granville. It is easy to determine whether a given integer is prime. *Bulletin American Math Society*, 42:3–38, 2005. Cited at 19, 135

[Gri01a]   D. Grigoriev. Complexity of Positivstellensatz proofs for the knapsack. *Computational complexity*, 10(2):139–154, 2001. Cited at 65, 235

[Gri01b]   D. Grigoriev. Linear lower bound on degrees of Positivstellensatz calculus proofs for the parity. *Theoretical Computer Science*, 259(1):613–622, 2001. Cited at 66, 235

[Gro87]   M. Gromov. Essays in group theory. *Mathematical Sciences Research Institute Publications*, 8:75–263, 1987. Cited at 20

[Gro96]   L.K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the 28th annual ACM symposium on Theory of Computing*, pages 212–219. ACM, 1996. Cited at 116

[Gro03]   M. Gromov. Random walk in random groups. *Geometric and Functional Analysis*, 13(1):73–146, 2003. Cited at 58, 140

[Gro10]   M. Gromov. Singularities, expanders and topology of maps. part 2: From combinatorics to topology via algebraic isoperimetry. *Geometric and Functional Analysis*, 20(2):416–526, 2010. Cited at 93

[Gro15]   J. A. Grochow. Unifying known lower bounds via geometric complexity theory. *Computational Complexity*, 24(2):393–475, 2015. Cited at 131

[GRS90]   R.L. Graham, B.L. Rothschild, and J.H. Spencer. *Ramsey Theory*, volume 20. John Wiley and Sons, New York, 1990. Cited at 82

[GRS16]   V. Guruswami, A. Rudra, and M. Sudan. Essential coding theory, 2016. http://www.cse.buffalo.edu/faculty/atri/courses/coding-theory/book/. Cited at 84

[GS71]   R.L. Graham and J.H. Spencer. A constructive solution to a tournament problem. *Canadian Math Bulletin*, 14(1):45–48, 1971. Cited at 84, 86

[GS89]   S. Goldwasser and M. Sipser. *Private coins versus public coins in interactive proof systems*, volume 5 of *Advances in Computing Research*. JAI Press, Inc., Greenwich, CT, 1989. Silvio Micali, ed. Cited at 103, 104, 105

[GS98]   V. Guruswami and M. Sudan. Improved decoding of Reed-Solomon and algebraic-geometric codes. In *Proceedings of 39th annual IEEE Symposium on Foundations of Computer Science*, pages 28–37. IEEE, 1998. Cited at 235

[GS00]   O. Goldreich and S. Safra. A combinatorial consistency lemma with application to proving the PCP theorem. *SIAM Journal on computing*, 29(4):1132–1154, 2000. Cited at 236

[GS06]   O. Goldreich and M. Sudan. Locally testable codes and PCPs of almost-linear length. *Journal of the ACM*, 53(4):558–655, 2006. Cited at 236

[GS14]   L. Gurvits and A. Samorodnitsky. Bounds on the permanent and some applications. In *Proceedings of 55th annual IEEE Symposium on Foundations of Computer Science*, pages 90–99. IEEE, 2014. Cited at 237

[GST14]   D. Genkin, A. Shamir, and E. Tromer. RSA key extraction via low-bandwidth acoustic cryptanalysis. In *International Cryptology Conference*, pages 444–461. Springer, 2014. Cited at 216, 217

[GT09]   B. Green and T. Tao. The distribution of polynomials over finite fields, with applications to the Gowers norms. *Contributions to Discrete Mathematics*, 4(2):1–36, 2009. Cited at 95

[GTZ12]   B. Green, T. Tao, and T. Ziegler. An inverse theorem for the Gowers $U^{s+1}[N]$-norm. *Annals of Mathematics*, 176(2):1231–1372, 2012. Cited at 96

[Gur08]   L. Gurvits. Van der Waerden/Schrijver-Valiant like conjectures and stable (aka hyperbolic) homogeneous polynomials: one theorem for all. *The electronic journal of combinatorics*, 15(1):R66, 2008. Cited at 139

[GUV09]   V. Guruswami, C. Umans, and S. Vadhan. Unbalanced expanders and randomness extractors from Parvaresh–Vardy codes. *Journal of the ACM*, 56(4):20, 2009. Cited at 101

[GV01]   D. Grigoriev and N. Vorobjov. Complexity of Null- and Positivstellensatz proofs. *Annals of Pure and Applied Logic*, 113(1):153–160, 2001. Cited at 65

[GV08]   A. Goel and V. Vogel. Harnessing biological motors to engineer systems for nanoscale transport and assembly. *Nature nanotechnology*, 3(8):465–475, 2008. Cited at 244

[GW96]   O. Goldreich and A. Wigderson. Theory of computation: A scientific perspective, 1996. `http://www.wisdom.weizmann.ac.il/~oded/toc-sp2.html`. Cited at 232

[GW11]   C. Gentry and D. Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In *Proceedings of the 43rd annual ACM symposium on Theory of Computing*, pages 99–108. ACM, 2011. Cited at 204

[GZ11]   O. Goldreich and D. Zuckerman. Another proof that BPP $\subseteq$ PH (and more). In *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation*, pages 40–53. Springer, 2011. Cited at 97

[Had10]   J. Hadamard. *Note sur quelques applications de l'indice de Kronecker*. Bussiere, 1910. Cited at 227

[Hae14]   B. Haeupler. Interactive channel capacity revisited. In *Proceedings of 55th annual IEEE Symposium on Foundations of Computer Science*, pages 226–235. IEEE, 2014. Cited at 177

[Hak61]   W. Haken. Theorie der Normalflächen: Ein Isotopiekriterium für den Kreisknoten. *Acta mathematica*, 105:245–375, 1961. Cited at 13

[Hak85]   A. Haken. The intractability of resolution. *Theoretical Computer Science*, 39:297–308, 1985. Cited at 67

[HAK07]   E. Hazan, A. Agarwal, and S. Kale. Logarithmic regret algorithms for online convex optimization. *Machine Learning*, 69(2-3):169–192, 2007. Cited at 184

[Hal05]   T.C. Hales. A proof of the Kepler conjecture. *Annals of mathematics*, pages 1065–1185, 2005. Cited at 14

[Ham50]   R.W. Hamming. Error detecting and error correcting codes. *Bell System technical journal*, 29(2):147–160, 1950. Cited at 176

[Har16]   D. Harlow. Jerusalem lectures on black holes and quantum information. *Reviews of Modern Physics*, 88(1):015002, 2016. Cited at 248

[Has86]   John Hastad. Almost optimal lower bounds for small depth circuits. In *Proceedings of the 18th annual ACM symposium on Theory of computing*, pages 6–20. Citeseer, 1986. Cited at 78

[Hås90]   J. Håstad. Tensor rank is NP-complete. *Journal of Algorithms*, 11(4):644–654, 1990. Cited at 133

[Hås98]   J. Håstad. The shrinkage exponent of de Morgan formulas is 2. *SIAM Journal on computing*, 27(1):48–64, 1998. Cited at 52

[Hås99]   J. Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. *Acta mathematica*, 182:105–142, 1999. Cited at 111

[Hås01]   J. Håstad. Some optimal inapproximability results. *Journal of the ACM*, 48:798–859, 2001. Cited at 111

[Has07]   M.B. Hastings. An area law for one-dimensional quantum systems. *Journal of Statistical Mechanics: Theory and Experiment*, 2007(08):P08024, 2007. Cited at 119

[Hat10]   H. Hatami. A structure theorem for Boolean functions with small total influences. *arXiv preprint arXiv:1008.1021*, 2010. Cited at 95

[Haz16]   E. Hazan. Introduction to online convex optimization. *Foundations and trends in optimization*, 2(3-4):157–325, 2016. Cited at 179

[HBD17]   H.A. Helfgott, J. Bajpai, and D. Dona. Graph isomorphisms in quasi-polynomial time. *arXiv preprint arXiv:1710.04574*, 2017. Cited at 20, 40, 103, 141

[HC99]   A. Haken and S.A. Cook. An exponential lower bound for the size of monotone real circuits. *Journal of computer and system sciences*, 58(2):326–335, 1999. Cited at 65

[Hea08]   A.D. Healy. Randomness-efficient sampling within NC^1. *Computational complexity*, 17(1):3–37, 2008. Cited at 101

[Hel08]   H.A. Helfgott. Growth and generation in $SL_2(\mathbb{Z}_p)$. *Annals of Mathematics*, pages 601–623, 2008. Cited at 92

[HEO05]   D.F. Holt, B. Eick, and E.A. O'Brien. *Handbook of computational group theory*. CRC Press, 2005. Cited at 141

[Her91]   M. Herlihy. Wait-free synchronization. *ACM Transactions on Programming Languages and Systems*, 13(1):124–149, 1991. Cited at 223, 224

[Het16]   S. Hetterich. Analysing survey propagation guided decimation on random formulas. *arXiv preprint arXiv:1602.08519*, 2016. Cited at 144

[HH13]   D. Harlow and P. Hayden. Quantum computation vs. firewalls. *Journal of High Energy Physics*, 6:085, 2013. Cited at 36, 248

[Hil93]   D. Hilbert. Über die vollen Invariantensysteme. *Mathematische Annalen*, 42(3):313–373, 1893. Cited at 150, 151

[HILL99] J. Håstad, R. Impagliazzo, L.A. Levin, and M. Luby. A pseudorandom generator from any one-way function. *SIAM Journal on computing*, 28(4):1364–1396, 1999. Cited at 78

[HJ16]   Y. Hu and H. Jia. Cryptanalysis of GGH map. In *Annual International conference on the Theory and Applications of Cryptographic Techniques*, pages 537–565. Springer, 2016. Cited at 216

[HKR13]   M. Herlihy, D. Kozlov, and S. Rajsbaum. *Distributed computing through combinatorial topology*. Newnes, 2013. Cited at 225, 230

[HL72]   O.J. Heilmann and E.H. Lieb. Theory of monomer-dimer systems. *Communications in Mathematical Physics*, 25(3):190–232, 1972. Cited at 142

[HLP99]   J. Hass, J.C. Lagarias, and N. Pippenger. The computational complexity of knot and link problems. *Journal of the ACM*, 46:185–211, 1999. Cited at 28, 32, 40

[HLW06]   S. Hoory, N. Linial, and A. Wigderson. Expander graphs and their applications. *Bulletin of the American Mathematical Society*, 43(4):439–561, 2006. Cited at 90, 91, 93, 138

[HMV06]   D. Hankerson, A.J. Menezes, and S. Vanstone. *Guide to elliptic curve cryptography*. Springer Science & Business Media, 2006. Cited at 217

[HO13]   B. Hemenway and R. Ostrovsky. Building lossy trapdoor functions from lossy encryption. In *International conference on the Theory and Application of Cryptology and Information Security*, pages 241–260. Springer, 2013. Cited at 211

[Hoa78]   C.A.R. Hoare. Communicating sequential processes. In *The origin of concurrent programming*, pages 413–443. Springer, 1978. Cited at 220

[Hru12]   P. Hrubeš. On the nonnegative rank of distance matrices. *Information Processing Letters*, 112(11):457–461, 2012. Cited at 170, 171

[HS65]   J. Hartmanis and R.E. Stearns. On the computational complexity of algorithms. *Transactions of the American Mathematical Society*, 117:285–306, 1965. Cited at 48

[HS99]   M. Herlihy and N. Shavit. The topological structure of asynchronous computability. *Journal of the ACM*, 46(6):858–923, 1999. Cited at 225, 226, 227, 229

[HS11]   M. Herlihy and N. Shavit. On the nature of progress. In *International Conference On Principles Of Distributed Systems*, pages 313–328. Springer, 2011. Cited at 220

[HS17]   S.B. Hopkins and D. Steurer. Bayesian estimation from few samples: community detection and related problems. *arXiv preprint arXiv:1710.00264*, 2017. Cited at 43, 259

[HSSW98]   D.P. Helmbold, R.E. Schapire, Y. Singer, and M.K. Warmuth. On-line portfolio selection using multiplicative updates. *Mathematical Finance*, 8(4):325–347, 1998. Cited at 184

[HT74]   J. Hopcroft and R. Tarjan. Efficient planarity testing. *Journal of the ACM*, 21(4):549–568, 1974. Cited at 19

[Huf52]   D. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 9(40):1098–1101, 1952. Cited at 174

[HW87]   D. Haussler and E. Welzl. ε-nets and simplex range queries. *Discrete & Computational Geometry*, 2(2):127–151, 1987. Cited at 194

[HW03]   J. Håstad and A. Wigderson. Simple analysis of graph tests for linearity and PCP. *Random Structures & Algorithms*, 22(2):139–160, 2003. Cited at 80

[HW14]   P. Hrubeš and A. Wigderson. Non-commutative arithmetic circuits with division. In *Proceedings of the 5th conference on innovations in theoretical Computer Science*, pages 49–66. ACM, 2014. Cited at 134

[HWY10] P. Hrubeš, A. Wigderson, and A. Yehudayoff. Non-commutative circuits and the sum-of-squares problem. In *Proceedings of the 42nd annual ACM symposium on Theory of Computing*, pages 667–676. ACM, 2010. Cited at 134

[HY11]   P. Hrubeš and A. Yehudayoff. Arithmetic complexity in algebraic extensions. *Theory of Computing*, 7(8):119–129, 2011. Cited at 125

[Hya79] L. Hyafil. On the parallel evaluation of multivariate polynomials. *SIAM Journal on computing*, 8(2):120–123, 1979. Cited at 126, 129

[IKW02] R. Impagliazzo, V. Kabanets, and A. Wigderson. In search of an easy witness: Exponential time vs. probabilistic polynomial time. *Journal of computer Systems and Sciences*, 65(4):672–694, 2002. Cited at 56, 81

[IKW12] R. Impagliazzo, V. Kabanets, and A. Wigderson. New direct-product testers and 2-query pcps. *SIAM Journal on computing*, 41(6):1722–1768, 2012. Cited at 236

[IL90]   R. Impagliazzo and L.A. Levin. No better ways to generate hard NP instances than picking uniformly at random. In *Proceedings of 31st annual IEEE Symposium on Foundations of Computer Science*, pages 812–821. IEEE, 1990. Cited at 43

[Imm88] N. Immerman. Nondeterministic space is closed under complementation. *SIAM Journal on computing*, 17(5):935–938, 1988. Cited at 155

[Imp95a] R. Impagliazzo. Hard-core distributions for somewhat hard problems. In *Proceedings of 36th annual IEEE Symposium on Foundations of Computer Science*, pages 538–545. IEEE, 1995. Cited at 95

[Imp95b] R. Impagliazzo. A personal view of average-case complexity. *Proceedings of the 10th annual IEEE conference on structure in complexity theory*, pages 134–147, 1995. Cited at 44

[Ind00] P. Indyk. Stable distributions, pseudorandom generators, embeddings and data stream computation. In *Proceedings of 41st annual IEEE Symposium on Foundations of Computer Science*, pages 189–197. IEEE, 2000. Cited at 157

[INW94] R. Impagliazzo, N. Nisan, and A. Wigderson. Pseudorandomness for network algorithms. In *Proceedings of the 26th annual ACM symposium on Theory of Computing*, pages 356–364. ACM, 1994. Cited at 89, 171, 172

[IP01]   R. Impagliazzo and R. Paturi. On the complexity of k-SAT. *Journal of computer and system sciences*, 62:367–375, 2001. Cited at 51, 235

[IPU94] R. Impagliazzo, T. Pitassi, and A. Urquhart. Upper and lower bounds for tree-like cutting planes proofs. In *Symposium on Logic in computer science, 1994.*, pages 220–228. IEEE, 1994. Cited at 168, 169

[IPZ01] R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *Journal of computer and system sciences*, 63(4):512–530, 2001. Cited at 51, 235

[IQS15] G. Ivanyos, Y. Qiao, and K.V. Subrahmanyam. Non-commutative Edmonds' problem and matrix semi-invariants. *arXiv preprint arXiv:1508.00690*, 2015. Cited at 153

[IV12]   T. Ito and T. Vidick. A multi-prover interactive proof for NEXP sound against entangled provers. In *Proceedings of 53rd annual IEEE Symposium on Foundations of Computer Science*, pages 243–252. IEEE, 2012. Cited at 121

[IW97]   R. Impagliazzo and A. Wigderson. P = BPP unless E has subexponential circuits: Deran-
           domizing the XOR lemma. In *Proceedings of the 29th annual ACM symposium on Theory of
           Computing*, pages 220–229. ACM, 1997. Cited at 72, 80, 100, 136

[IW98]   R. Impagliazzo and A. Wigderson. Randomness vs. time: De-randomization under a uniform
           assumption. In *Proceedings of 39th annual IEEE Symposium on Foundations of Computer
           Science*, pages 734–743, 1998. Cited at 73, 81

[IZ89]   R. Impagliazzo and D. Zuckerman. How to recycle random bits. In *Proceedings of 30th annual
           IEEE Symposium on Foundations of Computer Science*, pages 248–253. IEEE, 1989. Cited at
           101

[Jac94]   J. Jackson. An efficient membership-query algorithm for learning DNF with respect to the
           uniform distribution. In *Proceedings of 35th annual IEEE Symposium on Foundations of
           Computer Science*, pages 42–53. IEEE, 1994. Cited at 200

[Jer92]   M. Jerrum. Large cliques elude the metropolis process. *Random Structures & Algorithms*,
           3(4):347–359, 1992. Cited at 238

[Jer96]   M. Jerrum. The Markov chain Monte Carlo method: An approach to approximate counting
           and integration. *Approximation Algorithm for NP-hard Problems*, pages 482–520, 1996. Cited
           at 143, 237

[JJUW10]   R. Jain, Z. Ji, S. Upadhyay, and J. Watrous. QIP=PSPACE. *Communications of the ACM*,
           53(12):102–109, 2010. Cited at 121

[JP04]   N. C Jones and P. Pevzner. *An introduction to bioinformatics algorithms*. MIT Press, 2004.
           Cited at 244

[JS89]   M. Jerrum and A. Sinclair. Approximating the permanent. *SIAM journal on computing*,
           18(6):1149–1178, 1989. Cited at 237

[JS93]   M. Jerrum and G.B. Sorkin. Simulated annealing for graph bisection. In *Proceedings of 34th
           annual IEEE Symposium on Foundations of Computer Science*, pages 94–103. IEEE, 1993.
           Cited at 144, 238

[JSV04]   M. Jerrum, A. Sinclair, and E. Vigoda. A polynomial-time approximation algorithm for the
           permanent of a matrix with nonnegative entries. *Journal of the ACM*, 51(4):671–697, 2004.
           Cited at 72, 143, 237

[Juk12]   S. Jukna. *Boolean function complexity: advances and frontiers*, volume 27. Springer, 2012.
           Cited at 49

[Jus72]   J. Justesen. Class of constructive asymptotically good algebraic codes. *IEEE Transactions on
           Information Theory*, 18(5):652–656, 1972. Cited at 84

[JVV86]   M.R. Jerrum, L.G. Valiant, and V.V. Vazirani. Random generation of combinatorial structures
           from a uniform distribution. *Theoretical Computer Science*, 43:169–188, 1986. Cited at 143,
           237

[Kah74]   D. Kahn. *The codebreakers*. Weidenfeld and Nicolson, 1974. Cited at 207

[Kah95]   N. Kahale. Eigenvalues and expansion of regular graphs. *Journal of the ACM*, 42(5):1091–1106,
           1995. Cited at 91

[Kah96]   D. Kahn. *The Codebreakers: The comprehensive history of secret communication from ancient
           times to the internet*. Simon and Schuster, 1996. Cited at 202

[Kal83]   E. Kaltofen. Polynomial factorization. *Computer Algebra: Symbolic and Algebraic Computation, 2nd ed.*, 583:95–113, 1983. Cited at 72

[Kal85]   K.A. Kalorkoti. A lower bound for the formula size of rational functions. *SIAM Journal on computing*, 14(3):678–687, 1985. Cited at 129

[Kan12]   D.M. Kane. A structure theorem for poorly anticoncentrated Gaussian chaoses and applications to the study of polynomial threshold functions. In *Proceedings of 53rd annual IEEE Symposium on Foundations of Computer Science*, pages 91–100. IEEE, 2012. Cited at 95

[Kar72]   R. Karp. Reducibility among combinatorial problems. *Complexity of computer Computations*, pages 85–103, 1972. Cited at 31, 32, 33

[Kar76]   R.M. Karp. The probabilistic analysis of some combinatorial search algorithms. *Algorithms and complexity: New directions and recent results*, 1:19, 1976. Cited at 43, 259

[Kar84]   N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4:373–394, 1984. Cited at 19, 28, 235

[Kar11]   R.M. Karp. Understanding science through the computational lens. *Journal of Computer Science and Technology*, 26(4):569–577, 2011. Cited at 36

[Kha79]   L. Khachian. A polynomial time algorithm for linear programming. *Soviet Math. Doklady*, 10:191–194, 1979. Cited at 19, 28

[Kha93]   M. Kharitonov. Cryptographic hardness of distribution-specific learning. In *Proceedings of the 25th annual ACM symposium on Theory of Computing*, pages 372–381. ACM, 1993. Cited at 200

[Kho02]   S. Khot. On the power of unique 2-prover 1-round games. In *Proceedings of the 34th annual ACM symposium on Theory of Computing*, pages 767–775. ACM, 2002. Cited at 42, 43, 93, 140, 235

[Kho10]   S. Khot. Inapproximability of NP-complete problems, discrete Fourier analysis, and geometry. In *Proceedings of the International congress of mathematicians, vol. 5*, pages 2676–2697, New Delhi, 2010. Hindustan Book Agency. Cited at 43, 140

[Khr71]   V. M. Khrapchenko. A method of determining lower bounds for the complexity of π-schemes. *Mathematical Notes*, 10(1):474–479, 1971. Cited at 52

[KI04]   V. Kabanets and R. Impagliazzo. Identity tests means proving circuit lower bounds. *Computional Complexity*, 13(1–2):1–46, 2004. Cited at 80, 132

[Kil88]   J. Kilian. Founding cryptography on oblivious transfer. In *Proceedings of the 20th annual ACM symposium on Theory of Computing*, pages 20–31. ACM, 1988. Cited at 160

[Kit03]   A.Y. Kitaev. Fault-tolerant quantum computation by anyons. *Annals of Physics*, 303(1):2–30, 2003. Cited at 117, 118

[KKL88]   J. Kahn, G. Kalai, and N. Linial. The influence of variables on Boolean functions. In *Proceedings of 29th annual IEEE Symposium on Foundations of Computer Science*, pages 68–80. IEEE, 1988. Cited at 145

[KKMO07]   S. Khot, G. Kindler, E. Mossel, and R. O'Donnell. Optimal inapproximability results for MAX-CUT and other 2-variable CSPs? *SIAM Journal on computing*, 37(1):319–357, 2007. Cited at 145

[KL82]   R. Karp and R.J. Lipton. Turing machines that take advice. *L'Enseignement Mathematique*, 2(28):191–209, 1982. Cited at 50

[KL08]   T. Kaufman and S. Lovett. Worst case to average case reductions for polynomials. In *Proceedings of 49th annual IEEE Symposium on Foundations of Computer Science*, pages 166–175. IEEE, 2008. Cited at 95

[Kle51]   S.C. Kleene. Representation of events in nerve nets and finite automata. Technical report, RAND PROJECT AIR FORCE SANTA MONICA CA, 1951. Cited at 158

[Kle00]   J. Kleinberg. The small-world phenomenon: An algorithmic perspective. In *Proceedings of the 32nd annual ACM symposium on Theory of Computing*, pages 163–170. ACM, 2000. Cited at 252

[KLN06]   M. Kassabov, A. Lubotzky, and N. Nikolov. Finite simple groups as expanders. *Proceedings of the National Academy of Sciences*, 103(16):6116–6119, 2006. Cited at 92

[KLOS14]   J.A. Kelner, Y.T. Lee, L. Orecchia, and A Sidford. An almost-linear-time algorithm for approximate max flow in undirected graphs, and its multicommodity generalizations. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on discrete algorithms*, pages 217–226. SIAM, 2014. Cited at 235

[KLS01]   M. Kearns, M.L. Littman, and S. Singh. Graphical models for game theory. In *Proceedings of the 17th conference on uncertainty in artificial intelligence*, pages 253–260. Morgan Kaufmann Publishers Inc., 2001. Cited at 250

[KMRZS17]   S. Kopparty, O. Meir, N. Ron-Zewi, and S. Saraf. High-rate locally correctable and locally testable codes with sub-polynomial query complexity. *Journal of the ACM*, 64(2):11, 2017. Cited at 236

[KMS18]   S. Khot, D. Minzer, and M. Safra. Pseudorandom sets in Grassmann graph have near-perfect expansion. In *Electronic Colloquium on Computational Complexity (ECCC)*, volume 25, 2018. Cited at 43, 93

[KMSY14]   G. Kol, S. Moran, A. Shpilka, and A. Yehudayoff. Approximate nonnegative rank is equivalent to the smooth rectangle bound. In *International Colloquium on Automata, Languages, and Programming*, pages 701–712. Springer, 2014. Cited at 162

[KN97]   E. Kushilevitz and N. Nisan. *Communication Complexity*. Cambridge University Press, 1997. Cited at 161

[KO18]   T. Kaufman and I. Oppenheim. Construction of new local spectral high dimensional expanders. In *Proceedings of the 50th annual ACM Symposium on theory of Computing, 2018*, pages 773–786, 2018. Cited at 93

[Koi12]   P. Koiran. Arithmetic circuits: The chasm at depth four gets wider. *Theoretical Computer Science*, 448:56–65, 2012. Cited at 133

[KORW08]   G. Kindler, R. O'Donnell, A. Rao, and A. Wigderson. Spherical cubes and rounding in high dimensions. In *Proceedings of 49th annual IEEE Symposium on Foundations of Computer Science*, pages 189–198. IEEE, 2008. Cited at 146

[KP89]   J. Krajíček and P. Pudlák. Propositional proof systems, the consistency of first order theories and the complexity of computations. *The Journal of Symbolic Logic*, 54(03):1063–1079, 1989. Cited at 68

[KP95]   E. Koutsoupias and C.H. Papadimitriou. On the k-server conjecture. *Journal of the ACM*, 42(5):971–983, 1995. Cited at 181

[KP99]   E. Koutsoupias and C. Papadimitriou. Worst-case equilibria. In *Proceedings of the 16th annual conference on theoretical aspects of computer science, 1999*, pages 404–413. Springer, 1999. Cited at 250

[KPS85]   R. Karp, N. Pippenger, and M. Sipser. A time-randomness tradeoff. In *AMS conference on probabilistic computational complexity*, 1985. Cited at 92

[KPW92]   J. Komlós, J. Pach, and G. Woeginger. Almost tight bounds for ε-nets. *Discrete & Computational Geometry*, 7(2):163–173, 1992. Cited at 194

[KR13]   G. Kol and R. Raz. Interactive channel capacity. In *Proceedings of the 45th annual ACM symposium on Theory of Computing*, pages 715–724. ACM, 2013. Cited at 177

[Kra94]   J. Krajíček. Lower bounds to the size of constant-depth propositional proofs. *The Journal of Symbolic Logic*, 59(01):73–86, 1994. Cited at 68

[Kra95]   J Krajíček. *Bounded arithmetic, propositional logic and complexity theory*. Cambridge University Press, 1995. Cited at 57, 58

[Kra19]   J. Krajíček. *Proof complexity*. Cambridge University Press, 2019. Cited at 57

[Kri64]   J. Krivine. Anneaux préordonnés. *Journal d'analyse mathématique*, 12(1):307–326, 1964. Cited at 65

[KRR14]   Y.T. Kalai, R. Raz, and R.D. Rothblum. How to delegate computations: the power of no-signaling proofs. In *Proceedings of the 46th annual ACM symposium on Theory of Computing*, pages 485–494. ACM, 2014. Cited at 122, 215

[KRT17]   G. Kol, R. Raz, and A. Tal. Time-space hardness of learning sparse parities. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 1067–1080. ACM, 2017. Cited at 201

[KRVZ11]   J. Kamp, A. Rao, S. Vadhan, and D. Zuckerman. Deterministic extractors for small-space sources. *Journal of Computer and System Sciences*, 77(1):191–220, 2011. Cited at 102

[KRW95]   M. Karchmer, R. Raz, and A. Wigderson. Super-logarithmic depth lower bounds via the direct sum in communication complexity. *Computational complexity*, 5(3-4):191–204, 1995. Cited at 53, 54, 175

[KS59]   R.V. Kadison and I.M. Singer. Extensions of pure states. *American journal of mathematics*, 81(2):383–400, 1959. Cited at 138

[KS92]   B. Kalyanasundaram and G. Schintger. The probabilistic communication complexity of set intersection. *SIAM Journal on Discrete Mathematics*, 5(4):545–557, 1992. Cited at 163

[KS01]   A.R. Klivans and R. Servedio. Learning DNF in time $2^{\tilde{o}(n^{1/3})}$. In *Proceedings of the 33rd annual ACM symposium on Theory of Computing*, pages 258–265. ACM, 2001. Cited at 200

[KS09]   N. Kayal and S. Saraf. Blackbox polynomial identity testing for depth 3 circuits. In *Proceedings of 50th annual IEEE Symposium on Foundations of Computer Science*, pages 198–207. IEEE, 2009. Cited at 112, 138, 236

[KS13]   V. King and J. Saia. Byzantine agreement in polynomial expected time. In *Proceedings of the 45th annual ACM symposium on Theory of Computing*, pages 401–410. ACM, 2013. Cited at 224

[KS17]　　　P.K. Kothari and D. Steurer. Outlier-robust moment-estimation via sum-of-squares. *arXiv preprint arXiv:1711.11581*, 2017. Cited at 43, 259

[KSV02]　　　A.Y. Kitaev, A. Shen, and M.N. Vyalyi. *Classical and quantum computation*, volume 47. American Mathematical Society, 2002. Cited at 114, 118

[KSZ$^+$10]　　　S. Ko, M. Su, C. Zhang, A.E. Ribbe, W. Jiang, and C. Mao. Synergistic self-assembly of RNA and DNA molecules. *Nature chemistry*, 2(12):1050–1055, 2010. Cited at 244

[KT06]　　　J. Kleinberg and E. Tardos. *Algorithm design*. Pearson Education India, 2006. Cited at 18

[Kup14]　　　G. Kuperberg. Knottedness is in NP, modulo GRH. *Advances in Mathematics*, 256:493–506, 2014. Cited at 40

[KV94a]　　　M. Kearns and L.G. Valiant. Cryptographic limitations on learning Boolean formulae and finite automata. *Journal of the ACM*, 41(1):67–95, 1994. Cited at 200

[KV94b]　　　M.J. Kearns and U. Vazirani. *An introduction to computational learning theory*. MIT Press, 1994. Cited at 192

[KV03]　　　A. Kalai and S. Vempala. Efficient algorithms for universal portfolios. *The Journal of Machine Learning Research*, 3:423–440, 2003. Cited at 184

[KV05]　　　S.A. Khot and N.K. Vishnoi. The unique games conjecture, integrality gap for cut problems and embeddability of negative-type metrics into ℓ_1. In *Proceedings of 46th annual IEEE Symposium on Foundations of Computer Science*, pages 53–62. IEEE, 2005. Cited at 140

[KW90]　　　M. Karchmer and A. Wigderson. Monotone circuits for connectivity require super-logarithmic depth. *SIAM Journal on Discrete Mathematics*, 3(2):255–265, 1990. Cited at 52, 54, 55, 165, 166, 167

[KY06]　　　G. Kasparov and G. Yu. The coarse geometric Novikov conjecture and uniform convexity. *Advances in Mathematics*, 206(1):1–56, 2006. Cited at 140

[LAA87]　　　M.C. Loui and H.H. Abu-Amara. Memory requirements for agreement among unreliable asynchronous processes. *Advances in Computer Research*, 4:163–183, 1987. Cited at 224

[Lac15]　　　M. Lackenby. A polynomial upper bound on Reidemeister moves. *Annals of Mathematics*, 182:491–564, 2015. Cited at 28, 40, 58

[Lac16]　　　M. Lackenby. The efficient certification of knottedness and Thurston norm. *arXiv preprint arXiv:1604.00290*, 2016. Cited at 40

[Lad75]　　　R. Ladner. On the structure of polynomial time reducibility. *Journal of the ACM*, 22(1):155–171, 1975. Cited at 38, 39

[Laf08]　　　V. Lafforgue. Un renforcement de la propriété (T). *Duke Mathematical Journal*, 143(3):559–602, 2008. Cited at 140

[Lag84]　　　J.C. Lagarias. Knapsack public key cryptosystems and Diophantine approximation. In *Advances in cryptology*, pages 3–23. Springer, 1984. Cited at 148

[Lan12]　　　J.M. Landsberg. Tensors: geometry and applications. *Representation theory*, 381:402, 2012. Cited at 133

[Lan17]　　　J. Landsberg. *Geometry and complexity theory*. Cambridge University Press, 2017. Cited at 150

[Las01]   J.B. Lasserre. Global optimization with polynomials and the problem of moments. *SIAM Journal on Optimization*, 11(3):796–817, 2001. Cited at 65

[Las09]   J. Lasserre. *Moments, positive polynomials and their applications*, volume 1. World Scientific, 2009. Cited at 65

[Len83]   H.W. Lenstra. Integer programming with a fixed number of variables. *Mathematics of operations research*, 8(4):538–548, 1983. Cited at 148

[Lev73]   L. A. Levin. Universal search problems. *Problemy Peredachi Informatsii*, 9(3):115–116, 1973. English translation *Problems of Information Transmission* 9(3):265–266, 1973. Cited at 22, 30, 31

[Lev86]   L.A. Levin. Average case complete problems. *SIAM Journal on Computing*, 15(1):285–286, 1986. Cited at 43, 259

[Lev87]   L.A. Levin. One-way functions and pseudorandom generators. *Combinatorica*, 7(4):357–363, 1987. Cited at 47

[LFKN90]   C. Lund, L. Fortnow, H. Karloff, and N. Nisan. Algebraic methods for interactive proof systems. In *Proceedings of 31st annual IEEE Symposium on Foundations of Computer Science*, pages 2–10, 1990. Cited at 105

[Li16]   X. Li. Improved two-source extractors, and affine extractors for polylogarithmic entropy. In *Proceedings of 57th annual IEEE Symposium on Foundations of Computer Science*, pages 168–177. IEEE, 2016. Cited at 102

[Li17]   X. Li. Improved non-malleable extractors, non-malleable codes and independent source extractors. In *Proceedings of the 49th annual ACM symposium on Theory of Computing*, pages 1144–1156. ACM, 2017. Cited at 97

[Lin92]   N. Linial. Locality in distributed graph algorithms. *SIAM Journal on computing*, 21(1):193–201, 1992. Cited at 230, 231

[Lin02]   N. Linial. Finite metric spaces–combinatorics, geometry and algorithms. In *Proceedings of the International Congress of Mathematicians III*, pages 573–586. Citeseer, 2002. Cited at 139

[Lin17]   Y. Lindell. How to simulate it—a tutorial on the simulation proof technique. In *Tutorials on the Foundations of Cryptography*, pages 277–346. Springer, 2017. Cited at 207

[Lip91]   R. Lipton. New directions in testing. *DIMACS Distributed Computing and Cryptography, American Math Society*, 2:191–202, 1991. Cited at 106

[Lip94]   Richard J Lipton. Straight-line complexity and integer factorization. In *International Algorithmic Number Theory Symposium*, pages 71–79. 1994. Cited at 124

[Lip95]   R.J. Lipton. DNA solution of hard computational problems. *Science*, 268(5210):542, 1995. Cited at 244

[LLL82]   A.K. Lenstra, H.W. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, 1982. Cited at 19, 147, 148

[LLR95]   N. Linial, E. London, and Y. Rabinovich. The geometry of graphs and some of its algorithmic applications. *Combinatorica*, 15(2):215–245, 1995. Cited at 139

[LLS84]   R.E. Ladner, R.J. Lipton, and L.J. Stockmeyer. Alternating pushdown and stack automata. *SIAM Journal on computing*, 13(1):135–155, 1984. Cited at 159

[LM06]   N. Linial and R. Meshulam. Homological connectivity of random 2-complexes. *Combinatorica*, 26(4):475–487, 2006. Cited at 93

[LMM03]   R.J. Lipton, E. Markakis, and A. Mehta. Playing large games using simple strategies. In *Proceedings of the 4th ACM conference on electronic commerce*, pages 36–41. ACM, 2003. Cited at 250

[LMS11]   D. Lokshtanov, D. Marx, and S. Saurabh. Lower bounds based on the exponential time hypothesis. *Bulletin of the EATCS*, 3(105):41–72, 2011. Cited at 51

[LMSS01]   M.G. Luby, M. Mitzenmacher, M.A. Shokrollahi, and D.A. Spielman. Improved low-density parity-check codes using irregular graphs. *IEEE Transactions on Information Theory*, 47(2):585–598, 2001. Cited at 236

[LN15]   M. Lauria and J. Nordström. Tight size-degree bounds for sums-of-squares proofs. In *Proceedings of the 30th conference on computational complexity*, pages 448–466. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015. Cited at 66

[Lov12]   L. Lovász. *Large networks and graph limits*, volume 60. American Mathematical Society, 2012. Cited at 90, 240

[Lov14]   S. Lovett. Recent advances on the log-rank conjecture in communication complexity. *arXiv preprint arXiv:1403.8106*, 2014. Cited at 161, 162

[LP06]   A. Livnat and N. Pippenger. An optimal brain can be composed of conflicting agents. *Proceedings of the National Academy of Sciences*, 103(9):3198–3202, 2006. Cited at 246

[LP09]   L. Lovász and M.D. Plummer. *Matching theory*, volume 367. American Mathematical Society, 2009. Cited at 19

[LP16]   A. Livnat and C. Papadimitriou. Sex as an algorithm: the theory of evolution under the lens of computation. *Communications of the ACM*, 59(11):84–93, 2016. Cited at 246

[LPDF08]   A. Livnat, C. Papadimitriou, J. Dushoff, and M.W. Feldman. A mixability theory for the role of sex in evolution. *Proceedings of the National Academy of Sciences*, 105(50):19803–19808, 2008. Cited at 246

[LPPF10]   A. Livnat, C. Papadimitriou, N. Pippenger, and M.W. Feldman. Sex, mixability, and modularity. *Proceedings of the National Academy of Sciences*, 107(4):1452–1457, 2010. Cited at 246

[LPS88]   A. Lubotzky, R. Phillips, and P. Sarnak. Ramanujan graphs. *Combinatorica*, 8(3):261–277, 1988. Cited at 91, 92, 139

[LPS14]   J.W. Lichtman, H. Pfister, and N. Shavit. The big data challenges of connectomics. *Nature neuroscience*, 17(11):1448–1454, 2014. Cited at 247

[LR81]   D. Lehmann and M.O. Rabin. On the advantages of free choice: a symmetric and fully distributed solution to the dining philosophers problem. In *Proceedings of the 8th ACM SIGPLAN-SIGACT symposium on principles of programming languages*, pages 133–138. ACM, 1981. Cited at 221, 222, 224

[LRS14]   J.R. Lee, P. Raghavendra, and D. Steurer. Lower bounds on the size of semidefinite programming relaxations. *arXiv preprint arXiv:1411.6317*, 2014. Cited at 66, 235

[LRVW03]   C. Lu, O. Reingold, S. Vadhan, and A. Wigderson. Extractors: Optimal up to constant factors. In *Proceedings of the 35th annual ACM symposium on Theory of Computing*, pages 602–611. ACM, 2003. Cited at 137

[LS91]		L. Lovász and A. Schrijver. Cones of matrices and set-functions and 0-1 optimization. *SIAM Journal on Optimization*, 1(2):166–190, 1991. Cited at 65

[LS09]		T. Lee and A. Shraibman. Lower bounds in communication complexity. *Foundations and Trends in Theoretical Computer Science*, 3(4):263–399, 2009. Cited at 162

[LS14]		Y.T. Lee and A. Sidford. Path finding methods for linear programming: Solving linear programs in o(vrank) iterations and faster algorithms for maximum flow. In *Proceedings of 55th annual IEEE Symposium on Foundations of Computer Science*, pages 424–433. IEEE, 2014. Cited at 235

[LSP82]		L. Lamport, R. Shostak, and M. Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, 1982. Cited at 222, 223

[LSW00]		N. Linial, A. Samorodnitsky, and A. Wigderson. A deterministic strongly polynomial algorithm for matrix scaling and approximate permanents. *Combinatorica*, 20(4):545–568, 2000. Cited at 237

[LT89]		N.A. Lynch and M.R. Tuttle. An introduction to input/output automata. *CWI Quarterly*, 2(3):219–246, 1989. Cited at 222

[Lub02]		M. Luby. LT codes. In *Proceedings of 43rd annual IEEE Symposium on Foundations of Computer Science*, pages 271–280. IEEE, 2002. Cited at 236

[Lub14]		A. Lubotzky. Ramanujan complexes and high dimensional expanders. *Japanese Journal of Mathematics*, 9(2):137–169, 2014. Cited at 93

[LVV13]		Z. Landau, U. Vazirani, and T. Vidick. A polynomial-time algorithm for the ground state of 1d gapped local Hamiltonians. *arXiv preprint arXiv:1307.5143*, 2013. Cited at 120

[LW86]		N. Littlestone and M. Warmuth. Relating data compression and learnability. Technical report, University of California, Santa Cruz, 1986. Cited at 197

[LW92]		A. Lubotzky and B. Weiss. Groups and expanders. In Joel Friedman, editor, *Expanding graphs, Proceedings of a DIMACS Workshop*, pages 95–110. DIMACS/AMS, 1992. Cited at 92

[LW94]		N. Littlestone and M.K. Warmuth. The weighted majority algorithm. *Information and computation*, 108(2):212–261, 1994. Cited at 182, 183

[Lyn89]		N. Lynch. A hundred impossibility proofs for distributed computing. In *Proceedings of the 8th annual ACM symposium on principles of distributed computing*, pages 1–28. ACM, 1989. Cited at 218

[Lyn96]		N.A. Lynch. *Distributed algorithms*. Morgan Kaufmann, 1996. Cited at 218

[LZ77]		R.J. Lipton and Y. Zalcstein. Word problems solvable in Logspace. *Journal of the ACM*, 24(3):522–526, 1977. Cited at 154

[Mad13]		A. Madry. Navigating central path with electrical flows: From flows to matchings, and back. In *Proceedings of 54th annual IEEE Symposium on Foundations of Computer Science*, pages 253–262. IEEE, 2013. Cited at 235

[Mah18]		U. Mahadev. Classical verification of quantum computations. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science*, pages 259–267. IEEE, 2018. Cited at 121, 215

[Mal99]		J. Maldacena. The large N limit of superconformal field theories and supergravity. In *AIP conference proceedings CONF-981170*, volume 484, pages 51–63. AIP, 1999. Cited at 249

[Man80]		Y.I. Manin. *Vychislimoe i nevychislimoe (Computable and Noncomputable)*. Soviet Radio, 1980. In Russian. Cited at 114

[Mar73]		G.A. Margulis. Explicit constructions of concentrators. *Problems Information Transmission*, 9(4):71–80, 1973. Cited at 91, 92

[Mar88]		G.A. Margulis. Explicit group-theoretic constructions of combinatorial schemes and their applications in the construction of expanders and concentrators. *Problems Information Transmission*, 24:39–46, 1988. Cited at 91, 92, 139

[Mar06]		H. Markram. The blue brain project. *Nature Neuroscience*, 7(2):153, 2006. Cited at 247

[Mat02]		J. Matoušek. *Lectures on discrete geometry*, volume 108. Springer, 2002. Cited at 194

[Maz18]		A. Mazumdar. Capacity of locally recoverable codes. In *2018 IEEE Information Theory Workshop (ITW)*, pages 1–5. IEEE, 2018. Cited at 112

[McG14]		A. McGregor. Graph stream algorithms: a survey. *ACM SIGMOD Record*, 43(1):9–20, 2014. Cited at 156

[Mer97]		F. Mertens. Über eine zahlentheoretische Funktion. *Akademie Wissenschaftlicher Wien Mathematisch-Naturwissenschaftliche Klasse, Abteilung 2a*, 106:761–830, 1897. Cited at 85

[MFK82]		D. Mumford, J. Fogarty, and F. Kirwan. *Geometric invariant theory*, volume 34. Springer-Verlag, 2nd edition, 1982. Cited at 150

[Mil67]		S. Milgram. The small world problem. *Phychology Today*, 1:61–67, 1967. Cited at 252

[Mil76]		G.L. Miller. Riemann's hypothesis and tests for primality. *Journal of computer and system sciences*, 13(3):300–317, 1976. Cited at 28, 87, 136

[Min10]		H. Minkowski. *Geometrie der Zahlen*. Teubner, 1910. Cited at 147

[Mit97]		T. M. Mitchell. Does machine learning really work? *AI Magazine*, 18(3):11–20, 1997. Cited at 185

[MM11]		C. Moore and S. Mertens. *The nature of computation*. Oxford University Press, 2011. Cited at 5

[MMS90]		M.S. Manasse, L.A. McGeoch, and D.D. Sleator. Competitive algorithms for server problems. *Journal of Algorithms*, 11(2):208–230, 1990. Cited at 181

[MN14]		M. Mendel and A. Naor. Nonlinear spectral calculus and super-expanders. *Publications mathématiques de l'IHÉS*, 119(1):1–95, 2014. Cited at 140

[Moi16]		A. Moitra. Approximate counting, the Lovász local lemma and inference in graphical models. *arXiv preprint arXiv:1610.04317*, 2016. Cited at 144, 238

[MOO10]		E. Mossel, R. O'Donnell, and K. Oleszkiewicz. Noise stability of functions with low influences: Invariance and optimality. *Annals of Mathematics*, 171(1):295–341, 2010. Cited at 145, 146

[Mor85]		J. Morgenstern. How to compute fast a function and all its derivatives: A variation on the theorem of Baur-Strassen. *ACM SIGACT News*, 16(4):60–62, 1985. Cited at 127

[Mos09]		R.A. Moser. A constructive proof of the Lovász local lemma. In *Proceedings of the 41st annual ACM symposium on Theory of Computing*, pages 343–350. ACM, 2009. Cited at 144, 238

[MP43]		W.S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943. Cited at 158, 247

[MP69]   M. Minsky and S. Papert. Perceptrons. *MIT Press*, 18:19, 1969. Cited at 200

[MP15]   R. Meir and D. Parkes. On sex, evolution, and the multiplicative weights update algorithm. In *Proceedings of the 2015 international conference on autonomous agents and multiagent systems*, pages 929–937. International Foundation for Autonomous Agents and Multiagent Systems, 2015. Cited at 182, 246

[MPZ02]   M. Mézard, G. Parisi, and R. Zecchina. Analytic and algorithmic solution of random satisfiability problems. *Science*, 297(5582):812–815, 2002. Cited at 144

[MR95]   R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, Cambridge, 1995. Cited at 70, 72, 80

[MR04]   T. Mignon and N. Ressayre. A quadratic bound for the determinant and permanent problem. *International Mathematics Research Notices*, 2004(79):4241–4253, 2004. Cited at 131

[MR11]   E.W. Mayr and S. Ritscher. Space-efficient Gröbner basis computation without degree bounds. In *Proceedings of the 36th international symposium on symbolic and algebraic computation*, pages 257–264. ACM, 2011. Cited at 151

[MR13]   M. Mohri and A. Rostamizadeh. Perceptron mistake bounds. *arXiv preprint arXiv:1305.0208*, 2013. Cited at 187

[MS82]   K. Mehlhorn and E.M. Schmidt. Las Vegas is better than determinism in VLSI and distributed computing. In *Proceedings of the 14th annual ACM symposium on Theory of Computing*, pages 330–337. ACM, 1982. Cited at 162

[MS14a]   C.A. Miller and Y. Shi. Robust protocols for securely expanding randomness and distributing keys using untrusted quantum devices. In *Proceedings of the 46th annual ACM symposium on Theory of Computing*, pages 417–426. ACM, 2014. Cited at 123

[MS14b]   C. Moore and L.J. Schulman. Tree codes and a conjecture on exponential sums. In *Proceedings of the 5th conference on innovations in theoretical Computer Science*, pages 145–154. ACM, 2014. Cited at 178

[MS16]   C.A. Miller and Y. Shi. Robust protocols for securely expanding randomness and distributing keys using untrusted quantum devices. *Journal of the ACM (JACM)*, 63(4):33, 2016. Cited at 97, 123

[MSS13a]   A. Marcus, D.A. Spielman, and N. Srivastava. Interlacing families I: Bipartite Ramanujan graphs of all degrees. In *Proceedings of 54th annual IEEE Symposium on Foundations of Computer Science*, pages 529–537. IEEE, 2013. Cited at 93, 138, 139

[MSS13b]   A. Marcus, D.A. Spielman, and N. Srivastava. Interlacing families II: Mixed characteristic polynomials and the Kadison-Singer problem. *arXiv preprint arXiv:1306.3969*, 2013. Cited at 138, 139

[MT10]   R.A. Moser and G. Tardos. A constructive proof of the general Lovász local lemma. *Journal of the ACM*, 57(2):11, 2010. Cited at 144, 238

[Mul11]   K.D. Mulmuley. On P vs. NP and geometric complexity theory. *Journal of the ACM*, 58(2):5, 2011. Cited at 131, 150

[Mul12a]   K.D. Mulmuley. The GCT program toward the P vs. NP problem. *Communications of the ACM*, 55(6):98–107, 2012. Cited at 131, 150

[Mul12b] K.D. Mulmuley. Geometric complexity theory V: Equivalence between blackbox derandomization of polynomial identity testing and derandomization of Noether's normalization lemma. In *Proceedings of 53rd annual IEEE Symposium on Foundations of Computer Science*, pages 629–638. IEEE, 2012. Cited at 151, 152

[Mum95] D. Mumford. *Algebraic Geometry: Complex projective varieties*, volume 1. Springer Science & Business Media, 1995. Cited at 149

[Mur12] K.P. Murphy. *Machine learning: a probabilistic perspective*. MIT Press, 2012. Cited at 185

[Mut05] S. Muthukrishnan. *Data streams: Algorithms and applications*. Now Publishers Inc, 2005. Cited at 156

[MW04] R. Meshulam and A. Wigderson. Expanders in group algebras. *Combinatorica*, 24(4):659–680, 2004. Cited at 92

[MWW16] W. Mou, Z. Wang, and L. Wang. Stable memory allocation in the hippocampus: Fundamental limits and neural realization. *arXiv preprint arXiv:1612.04659*, 2016. Cited at 247

[MY16] S. Moran and A. Yehudayoff. Sample compression schemes for VC classes. *Journal of the ACM*, 63(3):21, 2016. Cited at 197

[Nao03] M. Naor. On cryptographic assumptions and challenges. In *Annual International Cryptology Conference*, pages 96–109. Springer, 2003. Cited at 204

[NC10] M.A. Nielsen and I.L. Chuang. *Quantum computation and quantum information*. Cambridge University Press, 2010. Cited at 114

[Nes00] Y. Nesterov. Squared functional systems and optimization problems. In *High performance optimization*, pages 405–440. Springer, 2000. Cited at 65

[New91] I. Newman. Private vs. common random bits in communication complexity. *Information processing letters*, 39(2):67–71, 1991. Cited at 162

[Nil91] A. Nilli. On the second eigenvalue of a graph. *Discrete Mathematics*, 91(2):207–210, 1991. Cited at 91

[Nis91a] N. Nisan. Lower bounds for non-commutative computation. In *Proceedings of the 23rd annual ACM symposium on Theory of Computing*, pages 410–418. ACM, 1991. Cited at 134

[Nis91b] N. Nisan. Pseudorandom bits for constant depth circuits. *Combinatorica*, 11(1):63–70, 1991. Cited at 79

[Nis92] N. Nisan. Pseudorandom generators for space-bounded computation. *Combinatorica*, 12(4):449–461, 1992. Cited at 88, 156, 171, 172

[Nis94] N. Nisan. RL ⊆ SC. *Computational complexity*, 4(1):1–11, 1994. Cited at 156

[Nis96] N. Nisan. Extracting randomness: how and why. A survey. In *Proceedings of the 11th annual IEEE conference on Computational complexity*, pages 44–58. IEEE, 1996. Cited at 97

[Nov62] A. Novikoff. On convergence proofs for perceptrons. In *Proceedings of the symposium on the mathematical theory of automata*, volume 12, pages 615–620. Polytechnic Press, 1962. Cited at 187, 191

[NRTV07] N. Nisan, T. Roughgden, E. Tardos, and V.V. Vazirani. *Algorithmic game theory*, volume 1. Cambridge University Press, 2007. Cited at 36, 250

[NW94]   N. Nisan and A. Wigderson. Hardness vs. randomness. *Journal of Computer and System Sciences*, 49(2):149–167, 1994. Cited at 79, 100

[NW96]   N. Nisan and A. Wigderson. Lower bounds on arithmetic circuits via partial derivatives. *Computational complexity*, 6(3):217–234, 1996. Cited at 134

[NY90]   M. Naor and M. Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *Proceedings of the 22nd annual ACM symposium on Theory of Computing*, pages 427–437. ACM, 1990. Cited at 207

[NY17]   A. Naor and R. Young. Vertical perimeter versus horizontal perimeter. *arXiv preprint arXiv:1701.00620*, 2017. Cited at 140

[NZ96]   N. Nisan and D. Zuckerman. Randomness is linear in space. *Journal of computer and system sciences*, 52(1):43–52, 1996. Cited at 99, 144

[O'D05]   R. O'Donnell. A history of the PCP theorem. *Course notes on the PCP Theorem and Hardness of Approximation*, 2005. Cited at 105

[O'D14]   R. O'Donnell. *Analysis of Boolean functions*. Cambridge University Press, 2014. Cited at 43, 145

[O'D17]   R. O'Donnell. SOS is not obviously automatizable, even approximately. In *8th Innovations in Theoretical Computer Science conference (ITCS 2017)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. Cited at 65

[Orl92]   A. Orlitsky. Average-case interactive communication. *IEEE Transactions on Information Theory*, 38(5):1534–1547, 1992. Cited at 174

[Orú14]   R. Orús. A practical introduction to tensor networks: Matrix product states and projected entangled pair states. *Annals of Physics*, 349:117–158, 2014. Cited at 119

[OtR85]   A. M. Odlyzko and H. J. J. te Riele. Disproof of the Mertens conjecture. *Journal für die reine und angewandte Mathematik*, 357:138–160, 1985. Cited at 148

[OW93]   R. Ostrovsky and A. Wigderson. One-way functions are essential for non-trivial zero-knowledge. In *[1993] The 2nd Israel Symposium on Theory and Computing Systems*, pages 3–17. IEEE, 1993. Cited at 107

[Pal33]   R.E. Paley. On orthogonal matrices. *Journal of Mathematics and Physics*, 12(1–4):311–320, 1933. Cited at 84

[PAM+10]   S. Pironio, A. Acín, S. Massar, A.. de la Giroday, D.N. Matsukevich, P. Maunz, S. Olmschenk, D. Hayes, L. Luo, T.A. Manning, and C. Monroe. Random numbers certified by Bell's theorem. *Nature*, 464(7291):1021–1024, 2010. Cited at 123

[Pap94]   C.H. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *Journal of Computer and System Sciences*, 48(3):498–532, 1994. Cited at 38, 59, 250

[Pap97]   C. H. Papadimitriou. NP-completeness: A retrospective. In *Proceedings of the 24th international colloquium on automata, languages and programming*, pages 2–6. Springer-Verlag, 1997. Cited at 33

[Pap03]   C.H. Papadimitriou. *Computational complexity*. John Wiley and Sons Ltd., 2003. Cited at 5

[Pap14]   C. Papadimitriou. Algorithms, complexity, and the sciences. *Proceedings of the National Academy of Sciences*, 111(45):15881–15887, 2014. Cited at 36

[Par00]   P.A. Parrilo. *Structured semidefinite programs and semialgebraic geometry methods in robust-ness and optimization.* PhD thesis, Citeseer, 2000. Cited at 65

[PB94]   P. Pudlák and S.R. Buss. How to lie without being (easily) convicted and the lengths of proofs in propositional calculus. In *International Workshop on Computer Science Logic*, volume 933, pages 151–162. Springer, 1994. Cited at 168

[Pei09]   C. Peikert. Public-key cryptosystems from the worst-case shortest vector problem. In *Proceedings of the 41st annual ACM symposium on Theory of Computing*, pages 333–342. ACM, 2009. Cited at 215

[Pei16]   C. Peikert. A decade of lattice cryptography. *Foundations and Trends in Theoretical Computer Science*, 10(4):283–424, 2016. Cited at 149

[Pev00]   P. Pevzner. *Computational molecular biology: an algorithmic approach.* MIT Press, 2000. Cited at 244

[Pig13]   G. Pighizzini. Two-way finite automata: Old and recent results. *Fundamenta Informaticae*, 126(2-3):225–246, 2013. Cited at 159

[Pin73]   M.S. Pinsker. On the complexity of a concentrator. In *7th International Telegraphic Confer-ence*, volume 4, pages 1–318. Citeseer, 1973. Cited at 90

[PPST83]   W.J. Paul, N. Pippenger, E. Szemeredi, and W.T. Trotter. On determinism versus non-determinism and related problems. In *Proceedings of 24th annual IEEE Symposium on Foun-dations of Computer Science*, pages 429–438. IEEE, 1983. Cited at 48

[PR17]   T. Pitassi and R. Robere. Strongly exponential lower bounds for monotone computation. In *Proceedings of the 49th annual ACM symposium on Theory of Computing*, pages 1246–1255. ACM, 2017. Cited at 54, 167

[Pro76]   C. Procesi. The invariant theory of $n \times n$ matrices. *Advances in Mathematics*, 19(3):306–381, 1976. Cited at 151

[Pud97]   P. Pudlak. Lower bounds for resolution and cutting planes proofs and monotone computations. *Journal of Symbolic Logic*, 62(3):981–998, 1997. Cited at 64

[Put93]   M. Putinar. Positive polynomials on compact semi-algebraic sets. *Indiana University Mathe-matics Journal*, 42(3):969–984, 1993. Cited at 65

[PV88]   L. Pitt and L.G. Valiant. Computational limitations on learning from examples. *Journal of the ACM*, 35(4):965–984, 1988. Cited at 200

[PV05]   F. Parvaresh and A. Vardy. Correcting errors beyond the Guruswami-Sudan radius in poly-nomial time. In *Proceedings of 46th annual IEEE Symposium on Foundations of Computer Science*, pages 285–294. IEEE, 2005. Cited at 101, 235

[PV15]   C.H. Papadimitriou and S. Vempala. Cortical learning via prediction. In *Conference on Learning Theory*, pages 1402–1422, 2015. Cited at 247

[PW85]   J. Paris and A. Wilkie. Counting problems in bounded arithmetic. In *Methods in mathematical logic*, pages 317–340. Springer, 1985. Cited at 58

[PW11]   C. Peikert and B. Waters. Lossy trapdoor functions and their applications. *SIAM Journal on computing*, 40(6):1803–1844, 2011. Cited at 47

[PZ89]   M. Pohst and H. Zassenhaus. *Algorithmic algebraic number theory.* Cambridge University Press, 1989. Cited at 147

[Rab63]   M.O. Rabin. Probabilistic automata. *Information and control*, 6(3):230–245, 1963. Cited at 159

[Rab67]   M. O. Rabin. Mathematical theory of automata. In Jacob T. Schwartz, editor, *Mathematical aspects of computer science, Proceedings of symposia in applied mathematics*, pages 153–175. American Mathematical Society, 1967. Cited at 17

[Rab79]   M.O. Rabin. Digitalized signatures and public-key functions as intractable as factorization. Technical report, DTIC Document, 1979. Cited at 206

[Rab80]   M.O. Rabin. Probabilistic algorithm for testing primality. *Journal of number theory*, 12(1):128–138, 1980. Cited at 71, 87, 136

[Rab81]   M.O. Rabin. How to exchange secrets with oblivious transfer. *Technical Memo TR-81*, 1981. Cited at 213

[RAD78]   R.L. Rivest, L. Adleman, and M.L. Dertouzos. On data banks and privacy homomorphisms. *Foundations of secure computation*, 4(11):169–180, 1978. Cited at 214

[Rag08]   P. Raghavendra. Optimal algorithms and inapproximability results for every CSP? In *Proceedings of the 40th annual ACM symposium on Theory of Computing*, pages 245–254. ACM, 2008. Cited at 42, 235

[Ram30]   F.P. Ramsey. On a problem of formal logic. *Proceedings of the London Mathematical Society*, 30:264–286, 1930. Cited at 82

[Raz74]   J.P. Razmyslov. Trace identities of full matrix algebras over a field of characteristic zero. *Mathematics of the USSR-Izvestiya*, 8(4):727, 1974. Cited at 151

[Raz85a]   A.A. Razborov. Lower bounds for the monotone complexity of some Boolean functions. *Dokl. Akad. Nauk SSSR*, 281(4):798–801, 1985. English transl. *Soviet Math. Doklady* **31** (1985), 354–357. Cited at 53

[Raz85b]   A.A. Razborov. Lower bounds on monotone complexity of the logical permanent. *Matematicheskie Zametki*, 37(6):887–900, 1985. Cited at 54

[Raz92]   A.A. Razborov. On the distributional complexity of disjointness. *Theoretical Computer Science*, 106(2):385–390, 1992. Cited at 163

[Raz95a]   A.A. Razborov. Bounded arithmetic and lower bounds in Boolean complexity. In *Feasible Mathematics II*, pages 344–386. Springer, 1995. Cited at 68

[Raz95b]   A.A. Razborov. Unprovability of lower bounds on the circuit size in certain fragments of bounded arithmetic. *Izvestiya of the Russian Academy of Science, Mathematics*, 59(1):201–224, 1995. Cited at 55, 68

[Raz96]   A.A. Razborov. Lower bounds for propositional proofs and independence results in bounded arithmetic. In *Automata, Languages and Programming*, pages 48–62. Springer, 1996. Cited at 68

[Raz98a]   R. Raz. A parallel repetition theorem. *SIAM Journal on computing*, 27(3):763–803, 1998. Cited at 146

[Raz98b]   A.A. Razborov. Lower bounds for the polynomial calculus. *Computational Complexity*, 7(4):291–324, 1998. Cited at 63

[Raz04a]   R. Raz. Resolution lower bounds for the weak pigeonhole principle. *Journal of the ACM*, 51(2):115–138, 2004. Cited at 69, 132

[Raz04b]   A.A. Razborov. Resolution lower bounds for perfect matching principles. *Journal of computer and system sciences*, 69(1):3–27, 2004. Cited at 69

[Raz11]   R. Raz. A counterexample to strong parallel repetition. *SIAM Journal on computing*, 40(3):771–777, 2011. Cited at 146

[Reg09]   O. Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM*, 56(6):34, 2009. Cited at 215

[Rei08]   O. Reingold. Undirected connectivity in log-space. *Journal of the ACM*, 55(4):17, 2008. Cited at 80, 92, 155

[Rem16]   Z. Remscrim. The Hilbert function, algebraic extractors, and recursive Fourier sampling. In *Proceedings of 57th annual IEEE Symposium on Foundations of Computer Science*, pages 197–208. IEEE, 2016. Cited at 102

[RM99]   R. Raz and P. McKenzie. Separation of the monotone NC hierarchy. *Combinatorica*, 19(3):403–435, 1999. Cited at 54

[Ros58]   F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958. Cited at 187

[Ros97]   A. Rosenbloom. Monotone real circuits are more powerful than monotone Boolean circuits. *Information Processing Letters*, 61(3):161–164, 1997. Cited at 65

[Rot53]   K.F. Roth. On certain sets of integers. *Journal of the London Mathematical Society*, 1(1):104–109, 1953. Cited at 94

[Rot06]   R. Roth. *Introduction to Coding Theory*. Cambridge University Press, 2006. Cited at 84

[Rot14]   T. Rothvoß. The matching polytope has exponential extension complexity. In *Proceedings of the 46th annual ACM symposium on Theory of Computing*, pages 263–272. ACM, 2014. Cited at 171

[Rou16]   T. Roughgarden. Communication complexity (for algorithm designers). *Foundations and Trends in Theoretical Computer Science*, 11(3–4):217–404, 2016. Cited at 161, 164

[RPRC16]   R. Robere, T. Pitassi, B. Rossman, and S. A. Cook. Exponential lower bounds for monotone span programs. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science*, pages 406–415. IEEE, 2016. Cited at 54, 167

[RPW04]   P.W.K. Rothemund, N. Papadakis, and E. Winfree. Algorithmic self-assembly of DNA Sierpinski triangles. *PLoS biology*, 2(12):e424, 2004. Cited at 244

[RR97]   A.A. Razborov and S. Rudich. Natural proofs. *Journal of computer and system sciences*, 55(1):24–35, 1997. Cited at 55, 86, 241

[RR99]   R. Raz and O. Reingold. On recycling the randomness of states in space bounded computation. In *Proceedings of the 31st annual ACM symposium on Theory of Computing*, pages 159–168. ACM, 1999. Cited at 97

[RRR16]   O. Reingold, G.N. Rothblum, and R.D. Rothblum. Constant-round interactive proofs for delegating computation. In *Proceedings of the 48th annual ACM symposium on Theory of Computing*, pages 49–62. ACM, 2016. Cited at 215

[RS59]   M.O. Rabin and D. Scott. Finite automata and their decision problems. *IBM journal of research and development*, 3(2):114–125, 1959. Cited at 158

[RS95]　　　N. Robertson and P. Seymour. Graph minors I–XIII. *Journal of Combinatorial Theory B*, 1983–1995. Cited at 20

[RS97]　　　R. Raz and S. Safra. A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP. In *Proceedings of the 29th annual ACM symposium on Theory of Computing*, pages 475–484. ACM, 1997. Cited at 236

[RS06]　　　V. Rödl and J. Skokan. Applications of the regularity lemma for uniform hypergraphs. *Random Structures & Algorithms*, 28(2):180–194, 2006. Cited at 96

[RS10]　　　P. Raghavendra and D. Steurer. Graph expansion and the unique games conjecture. In *Proceedings of the 42nd annual ACM symposium on Theory of Computing*, pages 755–764. ACM, 2010. Cited at 43

[RS11]　　　R. Rubinfeld and A. Shapira. Sublinear time algorithms. *SIAM Journal on Discrete Mathematics*, 25(4):1562–1588, 2011. Cited at 112

[RSA78]　　R.L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978. Cited at 206

[RSD16]　　O. Regev and N. Stephens-Davidowitz. A reverse Minkowski theorem. *arXiv preprint arXiv:1611.05979*, 2016. Cited at 147

[RSW04]　　E. Rozenman, A. Shalev, and A. Wigderson. A new family of Cayley expanders (?). In *Proceedings of the 36th annual ACM symposium on Theory of Computing*, pages 445–454. ACM, 2004. Cited at 92

[RT02]　　　T. Roughgarden and É. Tardos. How bad is selfish routing? *Journal of the ACM*, 49(2):236–259, 2002. Cited at 250

[RT18]　　　R. Raz and A. Tal. Oracle separation of BQP and PH. *Electronic Colloquium on Computational Complexity (ECCC)*, 25:107, 2018. Cited at 79, 116

[RTS00]　　J. Radhakrishnan and A. Ta-Shma. Bounds for dispersers, extractors, and depth-two superconcentrators. *SIAM Journal on Discrete Mathematics*, 13(1):2–24, 2000. Cited at 100

[RTTV08]　O. Reingold, L. Trevisan, M. Tulsiani, and S. Vadhan. Dense subsets of pseudorandom sets. In *Proceedings of 49th annual IEEE Symposium on Foundations of Computer Science*. IEEE, 2008. Cited at 95

[RTV06]　　O. Reingold, L. Trevisan, and S. Vadhan. Pseudorandom walks on regular digraphs and the RL vs. L problem. In *Proceedings of the 38th annual ACM symposium on Theory of Computing*, pages 457–466. ACM, 2006. Cited at 156

[RU01]　　　T.J. Richardson and R.L. Urbanke. The capacity of low-density parity-check codes under message-passing decoding. *IEEE Transactions on Information Theory*, 47(2):599–618, 2001. Cited at 236

[RVW02]　　O. Reingold, S. Vadhan, and A. Wigderson. Entropy waves, the zig-zag graph product, and new constant-degree expanders. *Annals of Mathematics*, pages 157–187, 2002. Cited at 92, 140

[RW89]　　　R. Raz and A. Wigderson. Probabilistic communication complexity of Boolean relations. In *Proceedings of 30th annual IEEE Symposium on Foundations of Computer Science*, pages 562–567. IEEE, 1989. Cited at 167

[RW92]　　　R. Raz and A. Wigderson. Monotone circuits for matching require linear depth. *J. ACM*, 39:736–744, 1992. Cited at 54, 166, 167

[RW00]   S. Rudich and A. Wigderson, editors. *Computational complexity theory*, volume 10 of *IAS/Park-City Mathematics Series*. Institute for Advanced Studies/American Math Society, 2000. Cited at 57

[Rys63]   H.J. Ryser. Combinatorial mathematics. *Math. Assoc. America*, 1963. Carus Mathematical Monographs, No. 14. Cited at 129

[SA90]   H.D. Sherali and W.P. Adams. A hierarchy of relaxations between the continuous and convex hull representations for zero-one programming problems. *SIAM Journal on Discrete Mathematics*, 3(3):411–430, 1990. Cited at 65

[San09]   R. Santhanam. Circuit lower bounds for Merlin-Arthur classes. *SIAM Journal on computing*, 39(3):1038–1061, 2009. Cited at 48, 56

[Sar90]   P. Sarnak. *Some applications of modular forms*, volume 99. Cambridge University Press, 1990. Cited at 91

[Sau72]   N. Sauer. On the density of families of sets. *Journal of Combinatorial Theory, Series A*, 13(1):145–147, 1972. Cited at 195

[Sav70]   W.J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of computer and system sciences*, 4(2):177–192, 1970. Cited at 106, 155

[Sch78]   T.J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the 10th annual ACM symposium on Theory of Computing*, pages 216–226. ACM, 1978. Cited at 41

[Sch80]   J.T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM*, 27(4):701–717, 1980. Cited at 70

[Sch90]   R.E. Schapire. The strength of weak learnability. *Machine learning*, 5(2):197–227, 1990. Cited at 197, 198

[Sch92]   L.J. Schulman. Communication on noisy channels: A coding theorem for computation. In *Proceedings of 33rd annual IEEE Symposium on Foundations of Computer Science*, pages 724–733. IEEE, 1992. Cited at 176, 254

[Sch93]   L.J. Schulman. Deterministic coding for interactive communication. In *Proceedings of the 25th annual ACM symposium on Theory of Computing*, pages 747–756. ACM, 1993. Cited at 176, 254

[Sch96]   L.J. Schulman. Coding for interactive communication. *IEEE Transactions on Information Theory*, 42(6):1745–1756, 1996. Cited at 176, 177

[Sch03]   A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Springer-Verlag, Berlin, 2003. Cited at 19

[Sch08]   G. Schoenebeck. Linear level Lasserre lower bounds for certain *k*-CSPs. In *Proceedings of 49th annual IEEE Symposium on Foundations of Computer Science*, pages 593–602. IEEE, 2008. Cited at 66

[See04]   N.C. Seeman. Nanotechnology and the double helix. *Scientific American*, 290(6):64–75, 2004. Cited at 244

[Seg07]   N. Segerlind. The complexity of propositional proofs. *Bulletin of Symbolic Logic*, 13(04):417–481, 2007. Cited at 57

[Sel65]   A. Selberg. On the estimation of Fourier coefficients of modular forms. In *Proceedings of Symposia in Pure Mathematics*, volume 8, pages 1–15, 1965. Cited at 91

[Ser03]   Á. Seress. *Permutation group algorithms*, volume 152. Cambridge University Press, 2003. Cited at 141

[SF12]   R.E.. Schapire and Y. Freund. *Boosting: Foundations and Algorithms*. The MIT Press, 2012. Cited at 197

[Sha48]   C.E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(3):379–423, 1948. Cited at 82, 83, 172, 173, 174, 176, 254

[Sha49a]   C.E. Shannon. Communication theory of secrecy systems. *Bell System Technical Journal*, 28(4):656–715, 1949. Cited at 203, 206

[Sha49b]   C.E. Shannon. The synthesis of two-terminal switching circuits. *Bell System Technical Journal*, 28(1):59–98, 1949. Cited at 50

[Sha79a]   A. Shamir. Factoring numbers in $O(\log n)$ arithmetic steps. *Information Processing Letters*, 8(1):28–31, 1979. Cited at 124

[Sha79b]   A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979. Cited at 213

[Sha83]   A. Shamir. On the generation of cryptographically strong pseudorandom sequences. *ACM Transactions on Computer Systems*, 1(1):38–44, 1983. Cited at 77

[Sha92]   A. Shamir. IP = PSPACE. *Journal of the ACM*, 39:869–877, 1992. Cited at 38, 105, 254

[Sha99]   Y. Shalom. Bounded generation and Kazhdan's property (T). *Publications Mathématiques de l'IHÉS*, 90:145–168, 1999. Cited at 92

[Sha04]   R. Shaltiel. Recent developments in explicit constructions of extractors. *Current Trends in Theoretical Computer Science: The Challenge of the New Century*, 1:229–264, 2004. Cited at 97

[She72]   S. Shelah. A combinatorial problem; stability and order for models and theories in infinitary languages. *Pacific Journal of Mathematics*, 41(1):247–261, 1972. Cited at 195

[She14]   A.A. Sherstov. Communication complexity theory: Thirty-five years of set disjointness. In *International symposium on Mathematical foundations of computer science*, pages 24–43. Springer, 2014. Cited at 161

[Sho88]   N.Z. Shor. An approach to obtaining global extremums in polynomial mathematical programming problems. *Cybernetics*, 23(5):695–700, 1988. Cited at 65

[Sho94]   P.W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings of 35th annual IEEE Symposium on Foundations of Computer Science*, pages 124–134. IEEE, 1994. Cited at 115, 137, 149

[Sho95]   P.W. Shor. Scheme for reducing decoherence in quantum computer memory. *Physical review A*, 52(4):R2493, 1995. Cited at 117

[Sim57]   H.A. Simon. *Models of man; social and rational*. Wiley, 1957. Cited at 243

[Sim70]   C.C. Sims. Computational methods in the study of permutation groups. In *Computational problems in abstract algebra*, pages 169–183, 1970. Cited at 20

[Sim97]   D.R. Simon. On the power of quantum computation. *SIAM Journal on computing*, 26(5):1474–1483, 1997. Cited at 116

[Sim10]   D. Simon. Selected applications of LLL in number theory. In *The LLL Algorithm*, pages 265–282. Springer, 2010. Cited at 148

[Sin11]   S. Singh. *The code book: the science of secrecy from ancient Egypt to quantum cryptography*. Anchor, 2011. Cited at 202

[Sip88]   M. Sipser. Expanders, randomness, or time versus space. *Journal of computer and system sciences*, 36(3):379–383, 1988. Cited at 98, 100

[Sip92]   M. Sipser. The history and status of the P versus NP question. In *Proceedings of the 24th annual ACM symposium on Theory of Computing*, pages 603–618. ACM, 1992. Cited at 23

[Sip97]   M. Sipser. *Introduction to the theory of computation*. PWS Publishing Co., Boston, MA, 1997. Cited at 158

[SJ89]   A. Sinclair and M. Jerrum. Approximate counting, uniform generation and rapidly mixing Markov chains. *Information and Computation*, 82(1):93–133, 1989. Cited at 90, 143, 237

[SK14]   S. Saraf and M. Kumar. On the power of homogeneous depth 4 arithmetic circuits. `http://www.math.rutgers.edu/~ss1984/`, 2014. Cited at 133

[Sly10]   A. Sly. Computational transition at the uniqueness threshold. In *Proceedings of 51st annual IEEE Symposium on Foundations of Computer Science*, pages 287–296. IEEE, 2010. Cited at 144, 238

[Spe28]   E. Sperner. Neuer beweis für die invarianz der dimensionszahl und des gebietes. In *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, volume 6, pages 265–272. Springer, 1928. Cited at 226

[Spi71]   P.M. Spira. On time-hardware complexity tradeoffs for Boolean functions. In *Proceedings of the 4th Hawaii symposium on system sciences*, pages 525–527, 1971. Cited at 166

[Spi95]   D.A. Spielman. Linear-time encodable and decodable error-correcting codes. In *Proceedings of the 27th annual ACM symposium on Theory of Computing*, pages 388–397. ACM, 1995. Cited at 176, 236

[SS71]   A. Schönhage and V. Strassen. Schnelle Multiplikation grosser Zahlen. *Computing*, 7:281–292, 1971. Cited at 51

[SS77]   R.M. Solovay and V. Strassen. A fast Monte-Carlo test for primality. *SIAM Journal on Computing*, 6(1):84–85, 1977. Cited at 71, 87, 136

[SS79]   E. Shamir and M. Snir. On the depth complexity of formulas. *Mathematical Systems Theory*, 13(1):301–322, 1979. Cited at 132

[SS96]   M. Sipser and D.A. Spielman. Expander codes. *IEEE Transactions on Information Theory*, 42(6):1710–1722, 1996. Cited at 236

[SS12]   D.A. Spielman and N. Srivastava. An elementary proof of the restricted invertibility theorem. *Israel Journal of Mathematics*, 190(1):83–91, 2012. Cited at 138

[ST85]   D.D. Sleator and R.E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985. Cited at 179, 181

[ST04a]   D.A. Spielman and S. Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the 36th annual ACM symposium on Theory of Computing*, pages 81–90. ACM, 2004. Cited at 235

[ST04b]   D.A. Spielman and S. Teng. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. *Journal of the ACM*, 51(3):385–463, 2004. Cited at 43, 235, 259

[ST11]   D.A. Spielman and S. Teng. Spectral sparsification of graphs. *SIAM Journal on computing*, 40(4):981–1025, 2011. Cited at 138

[Ste74]   G. Stengle. A Nullstellensatz and a Positivstellensatz in semialgebraic geometry. *Mathematische Annalen*, 207(2):87–97, 1974. Cited at 65

[STJ83]   E. Szemerédi and W.T. Trotter Jr. Extremal problems in discrete geometry. *Combinatorica*, 3(3-4):381–392, 1983. Cited at 138

[Sto73]   L.J. Stockmeyer. Planar 3-colorability is polynomial complete. *ACM Sigact News*, 5(3):19–25, 1973. Cited at 32, 33

[Sto76]   L.J. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):1–22, 1976. Cited at 37, 38

[Sto10]   J. Stothers. *On the Complexity of Matrix Multiplication*. PhD thesis, University of Edinburgh, 2010. Available at http://www.maths.ed.ac.uk/pg/thesis/stothers.pdf. Cited at 128

[Str69]   V. Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13(4):354–356, 1969. Cited at 128

[Str73a]   V. Strassen. Die Berechnungskomplexität von elementarsymmetrischen Funktionen und von Interpolationskoeffizienten. *Numerische Mathematik*, 20(3):238–251, 1973. Cited at 126

[Str73b]   V. Strassen. Vermeidung von Divisionen. *Journal für die reine und angewandte Mathematik*, 264:184–202, 1973. Cited at 127, 129

[Str86]   V. Strassen. The work of Leslie G. Valiant. In *International Congress of Mathematicians*, page 16, 1986. Cited at 35

[Str87]   V. Strassen. Relative bilinear complexity and matrix multiplication. *Journal für die reine und angewandte Mathematik*, 375:406–443, 1987. Cited at 150

[Stu08]   B. Sturmfels. *Algorithms in invariant theory*. Springer Science & Business Media, 2008. Cited at 149

[STV99]   M. Sudan, L. Trevisan, and S. Vadhan. Pseudorandom generators without the xor lemma. In *Proceedings of the 31st annual ACM symposium on Theory of Computing*, pages 537–546. ACM, 1999. Cited at 236

[Sub61]   B.A. Subbotovskaya. Realizations of linear functions by formulas using $+, \cdot, -$. *Doklady Akademii Nauk SSSR*, 136(3):553–555, 1961. Cited at 51

[Sud96]   M. Sudan. *Efficient Checking of Polynomials and Proofs and the Hardness of Approximation Problems*. Springer-Verlag, 1996. ACM Distinguished Theses, Lecture Notes in Computer Science. Cited at 110

[Sud97]   M. Sudan. Decoding of Reed-Solomon codes beyond the error-correction bound. *Journal of complexity*, 13(1):180–193, 1997. Cited at 235

[Sud00]   M. Sudan. List decoding: Algorithms and applications. *Theoretical Computer Science: Exploring New Frontiers of Theoretical Informatics*, pages 25–41, 2000. Cited at 236

[SV86]   M. Santha and U. Vazirani. Generating quasi-random sequences from semi-random sources. *Journal of Computer and System Sciences*, 33(1):75–87, 1986. Cited at 98

[SVdB01]   A. Schofield and M. Van den Bergh. Semi-invariants of quivers for arbitrary dimension vectors. *Indagationes Mathematicae*, 12(1):125–138, 2001. Cited at 152

[SW73]   D. Slepian and J. Wolf. Noiseless coding of correlated information sources. *IEEE Transactions on Information Theory*, 19(4):471–480, 1973. Cited at 174

[SW01]   A. Shpilka and A. Wigderson. Depth-3 arithmetic circuits over fields of characteristic zero. *Computational complexity*, 10(1):1–27, 2001. Cited at 126

[SW14]   A. Sahai and B. Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In *Proceedings of the 46th annual ACM symposium on Theory of Computing*, pages 475–484. ACM, 2014. Cited at 216

[Swa86]   E.R. Swart. P=NP. *Report No. CIS86-02, Department of Computer and Information Science, University of Guelph, Ontario, Canada*, 1986. Cited at 169

[SY10]   A. Shpilka and A. Yehudayoff. Arithmetic circuits: A survey of recent results and open questions. *Foundations and Trends in Theoretical Computer Science*, 5(3–4):207–388, 2010. Cited at 70, 124, 132, 134

[SZ99]   M. Saks and S. Zhou. $BP_H SPACE(S) \subseteq DSPACE(S^{3/2})$. *Journal of computer and system sciences*, 58(2):376–403, 1999. Cited at 156

[SZ00]   M. Saks and F. Zaharoglou. Wait-free k-set agreement is impossible: The topology of public knowledge. *SIAM Journal on computing*, 29(5):1449–1483, 2000. Cited at 225, 226, 227, 228

[Sze88]   R. Szelepcsényi. The method of forced enumeration for nondeterministic automata. *Acta Informatica*, 26(3):279–284, 1988. Cited at 155

[Sze12]   B. Szegedy. On higher order Fourier analysis. *arXiv preprint arXiv:1203.2260*, 2012. Cited at 96

[Tal14]   A. Tal. Shrinkage of de Morgan formulae by spectral techniques. In *Proceedings of 55th annual IEEE Symposium on Foundations of Computer Science*, pages 551–560. IEEE, 2014. Cited at 52

[Tan84]   R.M. Tanner. Explicit concentrators from generalized N-gons. *SIAM Journal on Algebraic Discrete Methods*, 5(3):287–293, 1984. Cited at 90

[Tao06]   T. Tao. The dichotomy between structure and randomness, arithmetic progressions, and the primes. In *Proceedings of the International Congress of Mathematicians*, pages 581–608, 2006. Cited at 93

[Tao08]   T. Tao. *Structure and randomness: pages from year one of a mathematical blog*. American Mathematical Society, Providence, RI, 2008. Cited at 93

[Tao09]   T. Tao. Recent progress on the Kakeya conjecture, 2009. `http://terrytao.wordpress.com/2009/05/11/`. Cited at 137

[Tar51]   A. Tarski. *A decision method for elementary algebra and geometry*. University of California Press, 1951. Cited at 13

[Tar87]   E. Tardos. The gap between monotone and non-monotone circuit complexity is exponential. *Combinatorica*, 7(4):141–142, 1987. Cited at 54

[Tav13]   S. Tavenas. Improved bounds for reduction to depth 4 and depth 3. In *Mathematical foundations of computer science 2013*, pages 813–824. Springer, 2013. Cited at 133

[Tel18]   R. Tell. Proving that prBPP=prP is as hard as "almost" proving that P $\neq$ NP. *Electronic Colloquium on Computational Complexity (ECCC)*, 25:3, 2018. Cited at 81

[Tho87]   Sir W. Thomson. On the division of space with minimum partitional area. *Acta mathematica*, 11(1-4):121–134, 1887. Cited at 146

[Tho79]   C.D. Thompson. Area-time complexity for VLSI. In *Proceedings of the 11th annual ACM symposium on Theory of Computing*, pages 81–88. ACM, 1979. Cited at 164

[Tho80]   C.D. Thompson. *A complexity theory for VLSI*. PhD thesis, Carnegie-Mellon University, 1980. Cited at 164

[Tho87]   A. Thomason. Pseudo-random graphs. *North-Holland Mathematics Studies*, 144:307–331, 1987. Cited at 89

[TKRR13]   Y. Tauman Kalai, R. Raz, and R.D. Rothblum. Delegation for bounded space. In *Proceedings of the 44th annual ACM symposium on Theory of Computing*, pages 565–574. ACM, 2013. Cited at 215

[Tod91]   S. Toda. PP is as hard as the polynomial-time hierarchy. *SIAM Journal on computing*, 20(5):865–877, 1991. Cited at 38, 106

[TPC15]   A. Tiwari, H.K. Patra, and J. Choi. *Advanced theranostic materials*. John Wiley & Sons, 2015. Cited at 244

[Tra84]   B.A. Trakhtenbrot. A survey of Russian approaches to perebor (brute-force searches) algorithms. *Annals of the History of Computing*, 6(4):384–400, 1984. Cited at 23

[Tre99]   L. Trevisan. Construction of extractors using pseudo-random generators. In *Proceedings of the 31st annual ACM symposium on Theory of Computing*, pages 141–148. ACM, 1999. Cited at 80, 100, 214

[Tsi93]   B. Tsirelson. Quantum Bell-type inequalities. *Hadronic Journal Supplement*, 8:329–345, 1993. Cited at 123

[TSZ04]   A. Ta-Shma and D. Zuckerman. Extractor codes. *IEEE Transactions on Information Theory*, 50(12):3015–3025, 2004. Cited at 97

[TT94]   P. Tiwari and M. Tompa. A direct version of Shamir and Snir's lower bounds on monotone circuit depth. *Information Processing Letters*, 49(5):243–248, 1994. Cited at 132

[TTV09]   L. Trevisan, M. Tulsiani, and S. Vadhan. Regularity, boosting, and efficiently simulating every high-entropy distribution. In *Proceedings of the 24th annual IEEE conference on Computational Complexity*, pages 126–136. IEEE, 2009. Cited at 95

[Tur36]   A.M Turing. On computable numbers, with an application to the Entscheidungsproblem. *Journal of Mathematics*, 58(345-363):5, 1936. Cited at 1, 12

[Tur50]   A.M. Turing. Computing machinery and intelligence. *Mind*, 59(236):433–460, 1950. Cited at 253

[Tur52]   A.M. Turing. The chemical basis of morphogenesis. *Philosophical Transactions of the Royal Society of London. Series B, Biological Sciences*, 237(641):37–72, 1952. Cited at 243

[TV00]   L. Trevisan and S. Vadhan. Extracting randomness from samplable distributions. In *Proceedings of 41st annual IEEE Symposium on Foundations of Computer Science*, page 32. IEEE, 2000. Cited at 102

[TV02]   L. Trevisan and S. Vadhan. Pseudorandomness and average-case complexity via uniform reductions. In *Proceedings of the 17th annual IEEE conference on computational complexity*, pages 129–138. IEEE, 2002. Cited at 81

[TZ08]   T. Tao and T. Ziegler. The primes contain arbitrarily long polynomial progressions. *Acta Mathematica*, 201(2):213–305, 2008. Cited at 95

[Vad04]   S.P. Vadhan. Constructing locally computable extractors and cryptosystems in the bounded-storage model. *Journal of Cryptology*, 17(1):43–77, 2004. Cited at 97

[Vad09]   S. Vadhan. Cryptographic applications of randomness extractors, 2009. `http://people.seas.harvard.edu/~salil/research/extractors-clouds09.ppt`. Cited at 97

[Vad11]   S.P. Vadhan. Pseudorandomness. *Foundations and Trends in Theoretical Computer Science*, 7(1–3):1–336, 2011. Cited at 70, 77, 97, 102, 137

[Vad17]   S. Vadhan. The complexity of differential privacy. In *Tutorials on the foundations of Cryptography*, pages 347–450. Springer, 2017. `http://privacytools.seas.harvard.edu/publications/complexity-differential-privacy`. Cited at 202

[Val79a] L.G. Valiant. Completeness classes in algebra. In *Proceedings of the 11th annual ACM symposium on Theory of Computing*, pages 249–261. ACM, 1979. Cited at 38, 130, 131

[Val79b] L.G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8(2):189–201, 1979. Cited at 144, 150, 237

[Val79c] L.G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on computing*, 8(3):410–421, 1979. Cited at 38, 237

[Val80]   L.G. Valiant. Negation can be exponentially powerful. *Theoretical Computer Science*, 12(3):303–314, 1980. Cited at 132

[Val84a] L.G. Valiant. Short monotone formulae for the majority function. *Journal of Algorithms*, 5(3):363–366, 1984. Cited at 55, 198

[Val84b] L.G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984. Cited at 192, 193, 195

[Val00]   L.G. Valiant. *Circuits of the Mind*. Oxford University Press on Demand, 2000. Cited at 36, 247

[Val06]   L.G. Valiant. A quantitative theory of neural computation. *Biological cybernetics*, 95(3):205–211, 2006. Cited at 247

[Val09]   L.G. Valiant. Evolvability. *Journal of the ACM*, 56(1):3, 2009. Cited at 245

[Val12]   L.G. Valiant. The hippocampus as a stable memory allocator for cortex. *Neural computation*, 24(11):2873–2899, 2012. Cited at 247

[Val13]   L.G. Valiant. *Probably Approximately Correct: Nature's Algorithms for Learning and Prospering in a Complex World*. Basic Books, 2013. Cited at 36, 192, 245

[Val14]   L.G. Valiant. What must a global theory of cortex explain? *Current opinion in neurobiology*, 25:15–19, 2014. Cited at 247

[Vap98]   V. Vapnik. *Statistical learning theory*, volume 1. Wiley, 1998. Cited at 192

[Vap13]   V. Vapnik. *The nature of statistical learning theory*. Springer Science & Business Media, 2013. Cited at 192

[Var57]   R.R. Varshamov. Estimate of the number of signals in error correcting codes. In *Doklady Akademia Nauk SSSR*, volume 117, pages 739–741, 1957. In Russian. English Translation in I. F. Blake, Algebraic Coding Theory: History and Development, Dowden, Hutchinson and Ross, 1973, pp. 68–71. Cited at 83, 176

[Vav09]   S.A. Vavasis. On the complexity of nonnegative matrix factorization. *SIAM Journal on Optimization*, 20(3):1364–1377, 2009. Cited at 170

[VC74]   V.N. Vapnik and A.J. Chervonenkis. *Theory of pattern recognition*. Nauka, 1974. Cited at 192

[VC15]   V.N. Vapnik and A.Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. In *Measures of Complexity*, pages 11–30. Springer, 2015. Cited at 192, 193, 194, 195

[Vem05]   S. Vempala. Geometric random walks: a survey. *Combinatorial and computational geometry*, 52(573–612):2, 2005. Cited at 144

[Vin04]   N.V. Vinodchandran. $AM_{exp} \not\subseteq (NP \cap coNP)/poly$. *Information Processing Letters*, 89:43–47, 2004. Cited at 48, 56

[Vio15]   E. Viola. The communication complexity of addition. *Combinatorica*, 35(6):703–747, 2015. Cited at 163

[vN28]   J. von Neumann. Zur theorie der gesellschaftsspiele. *Mathematische Annalen*, 100(1):295–320, 1928. Cited at 163

[vN51]   J. von Neumann. Various techniques used in connection with random digits. *Applied Math Series*, 12(36-38):1, 1951. Cited at 77

[VN58]   J. Von Neumann. *The computer and the brain*. Yale University Press, 1958. Reprinted in 2012 with a forward by Ray Kurzweil. Cited at 243, 247

[VNB⁺66]   J. Von Neumann, A.W. Burks, et al. Theory of self-reproducing automata. *IEEE Transactions on Neural Networks*, 5(1):3–14, 1966. Cited at 243

[VSBR83]   L.G. Valiant, S. Skyum, S. Berkowitz, and C. Rackoff. Fast parallel computation of polynomials using few processors. *SIAM Journal on computing*, 12(4):641–644, 1983. Cited at 126

[VV85]   U. Vazirani and V. Vazirani. Random polynomial time is equal to slightly-random polynomial time. In *Proceedings of 26th annual IEEE Symposium on Foundations of Computer Science*, pages 417–428. IEEE, 1985. Cited at 98

[VW18]   E. Viola and A. Wigderson. Local expanders. *Computational Complexity*, 27(2):225–244, 2018. Cited at 91

[vzGG13]   J. von zur Gathen and J. Gerhard. *Modern computer algebra*. Cambridge University Press, 2013. Cited at 124

[Wei49]   A. Weil. Numbers of solutions of equations in finite fields. *Bulletin Amer. Math. Soc.*, 55(5):497–508, 1949. Cited at 84, 85

[Wei06]   D. Weitz. Counting independent sets up to the tree threshold. In *Proceedings of the 38th annual ACM symposium on Theory of Computing*, pages 140–149. ACM, 2006. Cited at 144, 238

[Wer74]   P.J. Werbos. Beyond regression: New tools for prediction and analysis in the behavioral sciences. *Ph.D. dissertation, Harvard University*, 1974. Cited at 127

[Wer94]   P.J. Werbos. *The roots of backpropagation: from ordered derivatives to neural networks and political forecasting*, volume 1. John Wiley & Sons, 1994. Cited at 127

[Whi92]   S.R. White. Density matrix formulation for quantum renormalization groups. *Physical Review Letters*, 69(19):2863, 1992. Cited at 120

[Wig60]   E.P. Wigner. The unreasonable effectiveness of mathematics in the natural sciences. *Communications on pure and applied mathematics*, 13(1):1–14, 1960. Richard Courant lecture in mathematical sciences delivered at New York University, May 11, 1959. Cited at 242

[Wig93]   A. Wigderson. The fusion method for lower bounds in circuit complexity. *Combinatorics, Paul Erdos is Eighty*, 1:453–468, 1993. Cited at 41, 54, 56

[Wig06]   A. Wigderson. P, NP and mathematics—a computational complexity perspective. In *Proceedings of the 2006 International Congress of Mathematicians*, 2006. Cited at xiii

[Wig09]   A. Wigderson. Randomness extractors—applications and constructions. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 4. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2009. Cited at 97

[Wig10a]   A. Wigderson. Expander graphs—applications and combinatorial constructions, June 14–18 2010. A 3-hour tutorial, Pseudorandomness in Mathematical Structures Workshop. Available at `http://www.math.ias.edu/~avi/TALKS/Expander_tutorial_2010.ppt`. Cited at 91

[Wig10b]   A. Wigderson. The gödel phenomena in mathematics: A modern view. *Kurt Gödel and the Foundations of Mathematics: Horizons of Truth*, 2010. Cited at 23

[Wil75]   M.V. Wilkes. *Time sharing computer systems*. Elsevier Science Inc., 1975. Cited at 44

[Wil14]   R. Williams. Nonuniform ACC circuit lower bounds. *Journal of the ACM*, 61(1):2, 2014. Cited at 56

[Wil15]   V.V. Williams. Hardness of easy problems: Basing hardness on popular conjectures such as the strong exponential time hypothesis. In *Proceedings of the International symposium on Parameterized and Exact Computation*, pages 16–28, 2015. Cited at 235

[Wil18]   V.V. Williams. On some fine-grained questions in algorithms and complexity. In *Proceedings of the International Congress of Mathematicians (ICM)*, 2018. Cited at 51

[Win06]   E. Winfree. Self-healing tile sets. *Nanotechnology: science and computation*, pages 55–78, 2006. Cited at 244

[Wol99]   T. Wolff. Recent work connected with the Kakeya problem. *Prospects in mathematics*, 2:129–162, 1999. Cited at 137

[Yan91]   M. Yannakakis. Expressing combinatorial optimization problems by Linear Programs. *Journal of computer and system sciences*, 43(3):441–466, 1991. Cited at 169, 170, 235

[Yao77]   A.C. Yao. Probabilistic computations: Toward a unified measure of complexity. In *Proceedings of 18th annual IEEE Symposium on Foundations of Computer Science*, pages 222–227. IEEE, 1977. Cited at 163

[Yao79]   A.C. Yao. Some complexity questions related to distributive computing (preliminary report). In *Proceedings of the 11th annual ACM symposium on Theory of Computing*, pages 209–213. ACM, 1979. Cited at 161

[Yao82a]   A.C. Yao. Protocols for secure computations. In *Proceedings of 23rd annual IEEE Symposium on Foundations of Computer Science*, pages 160–164. IEEE, 1982. Cited at 211

[Yao82b]   A.C. Yao. Theory and application of trapdoor functions. In *Proceedings of 23rd annual IEEE Symposium on Foundations of Computer Science*, pages 80–91. IEEE, 1982. Cited at 73, 74, 77, 78, 207

[Yao86]   A.C. Yao. How to generate and exchange secrets. In *Proceedings of 27th annual IEEE Symposium on Foundations of Computer Science*, pages 162–167. IEEE, 1986. Cited at 46, 108, 209, 210, 211, 213, 251

[Yao93]   A.C. Yao. Quantum circuit complexity. In *Proceedings of 34th annual IEEE Symposium on Foundations of Computer Science*, pages 352–361. IEEE, 1993. Cited at 114

[Yeh11]   A. Yehudayoff. Affine extractors over prime fields. *Combinatorica*, 31(2):245, 2011. Cited at 102

[Yek12]   S. Yekhanin. Locally decodable codes. *Foundations and Trends in Theoretical Computer science*, 6(3):139–255, 2012. Cited at 112, 236

[Zhu17]   D. Zhuk. A proof of csp dichotomy conjecture. In *Proceedings of 58th annual IEEE Symposium on Foundations of Computer Science*, pages 331–342. IEEE, 2017. Cited at 41

[Zip79]   R.E. Zippel. Probabilistic algorithms for sparse polynomials. *Symbolic and algebraic computation (EUROSCAM '79), Lecture Notes in Computer Science*, 72:216–226, 1979. Cited at 70

[ZL78]   J. Ziv and A. Lempel. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, 24(5):530–536, 1978. Cited at 97

[Zuc90]   D. Zuckerman. General weak random sources. In *Proceedings of 31st Annual IEEE Symposium on Foundations of Computer Science*, pages 534–543. IEEE computer Society, 1990. Cited at 98, 100

[Zuc91]   D. Zuckerman. *Computing Efficiently Using General Weak Random Sources*. PhD thesis, U. C. Berkeley, 1991. Cited at 100

[Zuc97]   D. Zuckerman. Randomness-optimal oblivious sampling. *Random Structures and Algorithms*, 11(4):345–367, 1997. Cited at 102

[Zuc06]   D. Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. In *Proceedings of the 38th annual ACM symposium on Theory of Computing*, pages 681–690. ACM, 2006. Cited at 97