$\mathsf{P} \stackrel{?}{=} \mathsf{NP}$

Scott Aaronson*

Abstract

In 1955, John Nash sent a remarkable letter to the National Security Agency, in which—seeking to build theoretical foundations for cryptography—he all but formulated what today we call the $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ problem, considered one of the great open problems of science. Here I survey the status of this problem in 2016, for a broad audience of mathematicians, scientists, and engineers. I offer a personal perspective on what it's about, why it's important, why it's reasonable to conjecture that $\mathsf{P} \neq \mathsf{NP}$ is both true and provable, why proving it is so hard, the landscape of related problems, and crucially, what progress has been made in the last half-century toward solving those problems. The discussion of progress includes diagonalization and circuit lower bounds; the relativization, algebrization, and natural proofs barriers; and the recent works of Ryan Williams and Ketan Mulmuley, which (in different ways) hint at a duality between impossibility proofs and algorithms.

Contents

*University of Texas at Austin. Email: aaronson@cs.utexas.edu. Supported by a Vannevar Bush / NSSEFF Fellowship from the US Department of Defense. Much of this work was done while the author was supported by an NSF Alan T. Waterman award.

1 Introduction

> "Now my general conjecture is as follows: for almost all sufficiently complex types of enciphering, especially where the instructions given by different portions of the key interact complexly with each other in the determination of their ultimate effects on the enciphering, the mean key computation length increases exponentially with the length of the key, or in other words, the information content of the key ... The nature of this conjecture is such that I cannot *prove* it, even for a special type of ciphers. Nor do I expect it to be proven." —**John Nash, 1955** [189]

In 1900, David Hilbert challenged mathematicians to design a "purely mechanical procedure" to determine the truth or falsehood of any mathematical statement. That goal turned out to be impossible. But the *question*—does such a procedure exist, and why or why not?—helped launch two related revolutions that shaped the twentieth century: one in science and philosophy, as the results of Gödel, Church, Turing, and Post made the limits of reasoning itself a subject of mathematical analysis; and the other in technology, as the electronic computer achieved, not all of Hilbert's dream, but enough of it to change the daily experience of most people on earth.

Although there's no "purely mechanical procedure" to determine if a mathematical statement S is true or false, there *is* a mechanical procedure to determine if S has a proof of some bounded length n: simply enumerate over all proofs of length at most n, and check if any of them prove S. This method, however, takes exponential time. The $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ problem asks whether there's a *fast* algorithm to find such a proof (or to report that no proof of length at most n exists), for a suitable meaning of the word "fast." One can think of $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ as a modern refinement of Hilbert's 1900 question. The problem was explicitly posed in the early 1970s in the works of Cook and Levin, though versions were stated earlier—including by Gödel in 1956, and as we see above, by John Nash in 1955.

Think of a large jigsaw puzzle with (say) 10^{1000} possible ways of arranging the pieces, or an encrypted message with a similarly huge number of possible decrypts, or an airline with astronomically many ways of scheduling its flights, or a neural network with millions of weights that can be set independently. All of these examples share two key features:

(1) a finite but exponentially-large space of possible solutions; and

(2) a fast, mechanical way to check whether any claimed solution is "valid." (For example, do the puzzle pieces now fit together in a rectangle? Does the proposed airline schedule achieve the desired profit? Does the neural network correctly classify the images in a test suite?)

We're asking whether, under the above conditions, there's a general method to *find* a valid solution whenever one exists, and which is enormously faster than just trying all the possibilities one by one, from now till the end of the universe, like in Jorge Luis Borges' Library of Babel.

Notice that Hilbert's goal has been amended in two ways. On the one hand, the new task is "easier" because we've restricted ourselves to questions with only finitely many possible answers, each of which is easy to verify or rule out. On the other hand, the task is "harder" because we now insist on a *fast* procedure: one that avoids the exponential explosion inherent in the brute-force approach.

Of course, to discuss such things mathematically, we need to pin down the meanings of "fast" and "mechanical" and "easily checked." As we'll see, the $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ question corresponds to one natural choice for how to define these concepts, albeit not the only imaginable choice. For the impatient, P stands for "Polynomial Time," and is the class of all decision problems (that is, infinite sets of yes-or-no questions) solvable by a standard digital computer—or for concreteness, a Turing machine—using a polynomial amount of time. By "polynomial time," we mean that the machine uses a number of steps that's upper-bounded by the length of the input question (i.e., the number of bits needed to write it down) raised to some fixed power, as opposed (say) to growing exponentially with the length. Meanwhile, NP stands for "Nondeterministic Polynomial Time," and is the class of all decision problems for which, if the answer is "yes," then there's a polynomial-size proof that a Turing machine can *verify* in polynomial time. It's immediate that $\mathsf{P} \subseteq \mathsf{NP}$, so the question is whether this containment is proper (and hence $\mathsf{P} \neq \mathsf{NP}$), or whether $\mathsf{NP} \subseteq \mathsf{P}$ (and hence $\mathsf{P} = \mathsf{NP}$).

1.1 The Importance of $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$

Before getting formal, it seems appropriate to say something about the significance of the $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ question. $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$, we might say, shares with Hilbert's original question the character of a "math problem that's more than a math problem": a question that reaches inward to ask about mathematical reasoning itself, and also outward to everything from philosophy to natural science to practical computation.

To start with the obvious, essentially all the cryptography that we currently use on the Internet—for example, for sending credit card numbers—would be broken if $\mathsf{P} = \mathsf{NP}$ (and if, moreover, the algorithm were efficient in practice, a caveat we'll return to later). Though he was writing 21 years before $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ was explicitly posed, this is the point Nash was making in the passage with which we began.

The reason is that, in most cryptography, the problem of finding the decryption key is an NP search problem: that is, we know mathematically how to *check* whether a valid key has been found. The only exceptions are cryptosystems like the one-time pad and quantum key distribution, which don't rely on any computational assumptions (but have other disadvantages, such as the need for huge pre-shared keys or for special communication hardware).

The metamathematical import of $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ was also recognized early. It was articulated, for example, in Kurt Gödel's now-famous 1956 letter to John von Neumann, which sets out what we now call the $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ question. Gödel wrote:

> If there actually were a machine with [running time] $\sim Kn$ (or even only with $\sim Kn^2$) [for some constant K independent of n], this would have consequences of the greatest magnitude. That is to say, it would clearly indicate that, despite the unsolvability of

> the Entscheidungsproblem [Hilbert's problem of giving a complete decision procedure for mathematical statements], the mental effort of the mathematician in the case of yes-or-no questions could be completely [added in a footnote: apart from the postulation of axioms] replaced by machines. One would indeed have to simply select an n so large that, if the machine yields no result, there would then also be no reason to think further about the problem.

Expanding on Gödel's observation, some modern commentators have explained the importance of $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ as follows. It's well-known that $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ is one of the seven Clay Millennium Problems (alongside the Riemann Hypothesis, the Yang-Mills mass gap, etc.), for which a solution commands a million-dollar prize [72]. But even among those problems, $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ has a special status. For if someone discovered that $\mathsf{P} = \mathsf{NP}$, and if moreover the algorithm was efficient in practice, that person could solve not merely one Millennium Problem but *all seven of them*—for she'd simply need to program her computer to search for formal proofs of the other six conjectures.[1] Of course, if (as most computer scientists believe) $\mathsf{P} \neq \mathsf{NP}$, a proof of *that* would have no such world-changing implications, but even the fact that such a proof could *rule out* those implications underscores the enormity of what we're asking.

I should be honest about the caveats. While theoretical computer scientists (including me!) have not always been above poetic flourish, $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ is not quite equivalent to the questions of "whether human creativity can be automated," or "whether anyone who can appreciate a symphony is Mozart, anyone who can recognize a great novel is Jane Austen." Apart from the obvious point that *no* purely mathematical question could fully capture these imponderables, there are also more specific issues.

For one thing, while $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ has tremendous relevance to artificial intelligence, it says nothing about the *differences*, or lack thereof, between humans and machines. Indeed, $\mathsf{P} \neq \mathsf{NP}$ would represent a limitation on *all* classical digital computation, one that might plausibly apply to human brains just as well as to electronic computers. Nor does $\mathsf{P} \neq \mathsf{NP}$ rule out the possibility of robots taking over the world. To defeat humanity, presumably the robots wouldn't need to solve arbitrary NP problems in polynomial time: they'd merely need to be smarter than *us*, and to have imperfect heuristics better than the imperfect heuristics that *we* picked up from a billion years of evolution! Conversely, while a proof of $\mathsf{P} = \mathsf{NP}$ might hasten a robot uprising, it wouldn't guarantee one. For again, what $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ asks is not whether *all* creativity can be automated, but only *creativity whose fruits can be quickly verified by computer programs.*

To illustrate, suppose we wanted to program a computer to create new Mozart-quality symphonies and Shakespeare-quality plays. If $\mathsf{P} = \mathsf{NP}$ via a practical algorithm, then these feats reduce to the seemingly easier problem of writing a computer program to *recognize* great works of art. And interestingly, $\mathsf{P} = \mathsf{NP}$ might *also* help with the recognition problem: for example, by letting us train a neural network that reverse-engineered the expressed artistic preferences of hundreds of human experts. But how well that neural network would perform is an empirical question outside

[1] Here we're using the observation that, once we fix a formal system (say, first-order logic plus the axioms of ZF set theory), deciding whether a given statement has a proof at most n symbols long in that system is an NP problem, which can therefore be solved in time polynomial in n assuming $\mathsf{P} = \mathsf{NP}$. We're also assuming that the other six Clay conjectures have ZF proofs that are not too enormous: say, 10^{12} symbols or fewer, depending on the exact running time of the assumed algorithm. In the case of the Poincaré Conjecture, this can almost be taken to be a fact, modulo the translation of Perelman's proof [197] into the language of ZF.

the scope of mathematics.

1.2 Objections to $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$

After modest exposure to the $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ problem, many people come up with what they consider an irrefutable objection to its phrasing or importance. Since the same objections tend to recur, in this section I'll collect the most frequent ones and make some comments about them.

1.2.1 The Asymptotic Objection

Objection: $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ talks only about asymptotics—i.e., whether the running time of an algorithm grows polynomially or exponentially with the size n of the question that was asked, as n goes to infinity. It says nothing about the number of steps needed for concrete values of n (say, a thousand or a million), which is all anyone would ever care about in practice.

Response: It was realized early in the history of computer science that "number of steps" is not a robust measure of hardness, because it varies too wildly from one machine model to the next (from Macs to PCs and so forth), and also depends heavily on low-level details of how the problem is encoded. The asymptotic complexity of a problem could be seen as *that contribution to its hardness that is clean and mathematical*, and that survives the vicissitudes of technology. Of course, real-world software design requires thinking about many non-asymptotic contributions to a program's efficiency, from compiler overhead to the layout of the cache (as well as many considerations that have nothing to do with efficiency at all). But any good programmer knows that the asymptotics matter as well.

More specifically, many people object to theoretical computer science's identification of "polynomial" with "efficient" and "exponential" with "inefficient," given that for any practical value of n, an algorithm that takes 1.0000001^n steps is clearly preferable to an algorithm that takes n^{1000} steps. This would be a strong objection, if such algorithms were everyday phenomena. Empirically, however, computer scientists found that there *is* a strong correlation between "solvable in polynomial time" and "solvable efficiently in practice," with most (but not all) problems in P that they care about solvable in linear or quadratic or cubic time, and most (but not all) problems outside P that they care about requiring c^n time via any known algorithm, for some c significantly larger than 1. Furthermore, even when the first polynomial-time algorithm discovered for some problem takes (say) n^6 or n^{10} time, it often happens that later advances lower the exponent, or that the algorithm runs much faster in practice than it can be guaranteed to run in theory. This is what happened, for example, with linear programming, primality testing, and Markov Chain Monte Carlo algorithms.

Having said that, *of course* the goal is not just to answer some specific question like $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$, but to learn the truth about efficient computation, whatever it might be. If practically-important NP problems turn out to be solvable in n^{1000} time but not in n^{999} time, or in 1.0000001^n time, then so be it. From this perspective, one could argue that $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ simply serves as a marker of ignorance: in effect we're saying, "if we can't even answer *this*, then surely we can't answer the more refined questions either."

1.2.2 The Polynomial-Time Objection

Objection: But why should we draw the border of efficiency at the polynomial functions, as opposed to any other class of functions—for example, functions upper-bounded by n^2, or functions of the form $n^{\log^c n}$ (called *quasipolynomial* functions)?

Response: There's a good theoretical answer to this: it's because polynomials are the smallest class of functions that contain the linear functions, and that are closed under basic operations like addition, multiplication, and composition. For this reason, they're the smallest class that ensures that we can compose "efficient algorithms" a constant number of times, and still get an algorithm that's efficient overall. For the same reason, polynomials are *also* the smallest class that ensures that our "set of efficiently solvable problems" is independent of the low-level details of the machine model.

Having said that, much of algorithms research *is* about lowering the order of the polynomial, for problems already known to be in P, and theoretical computer scientists *do* use looser notions like quasipolynomial time whenever they're needed.

1.2.3 The Kitchen-Sink Objection

Objection: $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ is limited, because it talks only about discrete, deterministic algorithms that find exact solutions in the worst case—and also, because it ignores the possibility of natural processes that might exceed the limits of Turing machines, such as analog computers, biological computers, or quantum computers.

Response: For every assumption mentioned above, there's now a major branch of theoretical computer science that studies what happens when one relaxes the assumption: for example, randomized algorithms, approximation algorithms, average-case complexity, and quantum computing. I'll discuss some of these branches in Section 5. Briefly, though, there are deep reasons why many of these ideas are thought to leave the original $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ problem in place. For example, according to the $\mathsf{P} = \mathsf{BPP}$ conjecture (see Section 5.4.1), randomized algorithms yield no more power than P, while careful analyses of noise, energy expenditure, and the like suggest that the same is true for analog computers (see [3]). Meanwhile, the famous *PCP Theorem* and its offshoots (see Section 3) have shown that, for many NP problems, there can't even be a polynomial-time algorithm to *approximate* the answer to within a reasonable factor, unless $\mathsf{P} = \mathsf{NP}$.

In other cases, new ideas have led to major, substantive *strengthenings* of the $\mathsf{P} \neq \mathsf{NP}$ conjecture (see Section 5): for example, that there exist NP problems that are hard even on random inputs, or hard even for a quantum computer. Obviously, proving $\mathsf{P} \neq \mathsf{NP}$ itself is a prerequisite to proving any of these strengthened versions.

There's one part of this objection that's so common that it requires some separate comments. Namely, people say that even if $\mathsf{P} \neq \mathsf{NP}$, *in practice* we can almost always find good enough solutions to the problems we care about, for example by using heuristics like simulated annealing or genetic algorithms, or by using special structure or symmetries in real-life problem instances.

Certainly there are cases where this assumption is true. But there are also cases where it's false: indeed, the entire field of cryptography is about *making* the assumption false! In addition, I believe our practical experience is biased by the fact that we don't even *try* to solve search problems that we "know" are hopeless—yet that wouldn't be hopeless in a world where $\mathsf{P} = \mathsf{NP}$ (and where the algorithm was efficient in practice). For example, presumably no one would try using brute-force search to look for a formal proof of the Riemann Hypothesis one billion lines long or shorter, or a

10-megabyte program that reproduced most of the content of Wikipedia within a reasonable time (possibly needing to encode many of the principles of human intelligence in order to do so). Yet both of these are "merely" NP search problems, and things one could seriously contemplate in a world where P = NP.

1.2.4 The Mathematical Snobbery Objection

Objection: $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ is not a "real" math problem, because it talks about Turing machines, which are arbitrary human creations, rather than about "natural" mathematical objects like integers or manifolds.

Response: The simplest reply is that $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ is not about Turing machines at all, but about *algorithms*, which seem every bit as central to mathematics as integers or manifolds. Turing machines are just one particular formalism for expressing algorithms, as the Arabic numerals are one formalism for integers. And just like the Riemann Hypothesis is still the Riemann Hypothesis in base-17 arithmetic, so essentially *every* formalism for deterministic digital computation ever proposed gives rise to the same complexity classes P and NP, and the same question about whether they're equal. (This observation is known as the *Extended Church-Turing Thesis*.)

This objection might also reflect lack of familiarity with recent progress in complexity theory, which has drawn on Fourier analysis, arithmetic combinatorics, representation theory, algebraic geometry, and dozens of other subjects about which yellow books[2] are written. Furthermore, in Section 6.6, we'll see Geometric Complexity Theory (GCT): a staggeringly ambitious program for proving $\mathsf{P} \neq \mathsf{NP}$ that throws almost the entire arsenal of modern mathematics at the problem, including geometric invariant theory, plethysms, quantum groups, and Langlands-type correspondences—and that relates $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$, at least conjecturally, to other questions that mathematicians have been trying to answer for a century. Even if GCT's specific conjectures don't pan out, they illustrate how progress toward proving $\mathsf{P} \neq \mathsf{NP}$ could involve deep insights from many parts of mathematics.

1.2.5 The Sour Grapes Objection

Objection: $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ is *so* hard that it's impossible to make anything resembling progress on it, at least at this stage in human history—and for that reason, it's unworthy of serious effort or attention. Indeed, we might as well treat such questions as if their answers were formally independent of set theory, as for all we know they are (a possibility discussed further in Section 3.1).

Response: One of the main purposes of this survey is to explain what we know now, relevant to the $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ problem, that we didn't know 10 or 20 or 30 years ago. It's true that, if "progress" entails having a solution already in sight, or being able to estimate the time to a solution, I know of no progress of *that* kind! But by the same standard, one would have to say there was no "progress" toward Fermat's Last Theorem in 1900—even as mathematicians, partly motivated by Fermat's problem, were laying foundations of algebraic number theory that *did* eventually lead to Wiles's proof. In this survey, I'll try to convey how, over the last few decades, insights about circuit lower bounds, relativization and arithmetization, pseudorandomness and natural proofs, the "duality" between lower bounds and algorithms, the permanent and determinant manifolds, and more have transformed our understanding of what a $\mathsf{P} \neq \mathsf{NP}$ proof could look like.

[2]Because I was asked: "yellow books" are the Springer mathematics books that line many mathematicians' offices.

I should point out that, even supposing $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ is *never* solved, it's already been remarkably fruitful as an "aspirational" or "flagship" question, helping to shape research in algorithms, cryptography, learning theory, derandomization, quantum computing, and other areas that theoretical computer scientists work on. Furthermore, later we'll see examples of how progress in some of those other areas unexpectedly ended up tying back to the quest to prove $\mathsf{P} \neq \mathsf{NP}$.

1.2.6 The Obviousness Objection

Objection: It's intuitively obvious that $\mathsf{P} \neq \mathsf{NP}$. For that reason, a proof of $\mathsf{P} \neq \mathsf{NP}$—confirming that indeed, we can't do something that no reasonable person would ever have imagined we could do—gives almost no useful information.

Response: This objection is perhaps less common among mathematicians than others, since were it upheld, it would generalize to *almost all* of mathematics! Like with most famous unsolved math problems, the quest to prove $\mathsf{P} \neq \mathsf{NP}$ is "less about the destination than the journey": there might or might not be surprises in the answer itself, but there will *certainly* be huge surprises (indeed, there have already been huge surprises) along the way. More concretely: to make a sweeping statement like $\mathsf{P} \neq \mathsf{NP}$, about what polynomial-time algorithms *can't* do, will require an unprecedented understanding of what they *can* do. This will almost certainly entail the discovery of many new polynomial-time algorithms, some of which could have practical relevance. In Section 6, we'll see much more subtle examples of the "duality" between algorithms and impossibility proofs, with progress on each informing the other.

Of course, to whatever extent you regard $\mathsf{P} = \mathsf{NP}$ as a live possibility, the Obviousness Objection isn't open to you.

1.2.7 The Constructivity Objection

Objection: Even if $\mathsf{P} = \mathsf{NP}$, the proof could be nonconstructive—in which case it wouldn't have any of the amazing implications discussed in Section 1.1, because we wouldn't know the algorithm.

Response: A nonconstructive proof that an algorithm exists is indeed a theoretical possibility, though one that's reared its head only a few times in the history of computer science.[3] Even then, however, once we knew that an algorithm *existed*, we'd have a massive inducement to try to find it. The same is true if, for example, the first proof of $\mathsf{P} = \mathsf{NP}$ only gave an n^{1000} algorithm, but we suspected that an n^2 algorithm existed.[4]

[3] The most celebrated examples of nonconstructive proofs that algorithms exist all come from the *Robertson-Seymour graph minors theory*, one of the great achievements of 20^{th}-century combinatorics (for an accessible introduction, see for example Fellows [82]). The Robertson-Seymour theory typically deals with *parameterized* problems: for example, "given a graph G, decide whether G can be embedded on a sphere with k handles." In those cases, typically a fast algorithm A_k can be abstractly shown to exist for every value of k. The central problem is that each A_k requires hard-coded data—in the above example, a finite list of obstructions to the desired embedding—that no one knows how to find given k, and whose size might also grow astronomically as a function of k. On the other hand, once the finite obstruction set for a given k was known, one could then use it to solve the problem for any graph G in time $O\left(|G|^3\right)$, where the constant hidden by the big-O depended on k.

Robertson-Seymour theory also provides a few examples of non-parameterized problems that are abstractly proved to be in P but with no bound on the exponent, or abstractly proved to be $O\left(n^3\right)$ or $O\left(n^2\right)$ but with no bound on the constant. Thus, one can't rule out the possibility that the same would happen with an NP-complete problem, and Donald Knuth [144] has explicitly speculated that $\mathsf{P} = \mathsf{NP}$ will be proven in that way. To me, however, it's unclear whether he speculates this because there's a positive reason for thinking it true, or just because it would be cool and interesting if it *were* true.

[4] As an amusing side note, there's a trick called *Levin's universal search* [157], in which one "dovetails" over all

1.3 Further Reading

There were at least four previous major survey articles about $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$: Michael Sipser's 1992 "The History and Status of the P versus NP Question" [227]; Stephen Cook's 2000 "The P versus NP Problem" [72], which was written for the announcement of the Clay Millennium Prize; Avi Wigderson's 2006 "P, NP, and Mathematics—A Computational Complexity Perspective" [254]; and Eric Allender's 2009 "A Status Report on the P versus NP Question" [22]. All four are excellent, so it's only with trepidation that I add another entry to the crowded arena. I hope that, if nothing else, this survey shows how much has continued to occur through 2016. I cover several major topics that either didn't exist a decade ago, or existed only in much more rudimentary form: for example, the algebrization barrier, "ironic complexity theory" (including Ryan Williams's $\mathsf{NEXP} \not\subset \mathsf{ACC}$ result), the "chasm at depth three" for the permanent, and the Mulmuley-Sohoni Geometric Complexity Theory program.

The seminal papers that set up the intellectual framework for $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$, posed it, and demonstrated its importance include those of Edmonds [81], Rabin [199], Cobham [71], Cook [73], Karp [134], and Levin [157]. See also Trakhtenbrot [244] for a survey of Soviet thought about *perebor*, as brute-force search was referred to in Russian in the 1950s and 60s.

The classic text that introduced the wider world to P, NP, and NP-completeness, and that gave a canonical (and still-useful) list of hundreds of NP-complete problems, is Garey and Johnson [96]. Some recommended computational complexity theory textbooks—in rough order from earliest to most recent, in the material they cover—are Sipser [228], Papadimitriou [193], Schöning [219], Moore and Mertens [172], and Arora and Barak [28]. Surveys on particular aspects of complexity theory will be recommended where relevant throughout the survey.

Those seeking a nontechnical introduction to $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ might enjoy Lance Fortnow's charming book *The Golden Ticket* [88], or his 2009 popular article for *Communications of the ACM* [87]. My own *Quantum Computing Since Democritus* [6] gives something between a popular and a technical treatment.

2 Formalizing $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ and Central Related Concepts

The $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ problem is normally phrased in terms of *Turing machines*: a theoretical model of computation proposed by Alan Turing in 1936, which involves a one-dimensional tape divided into discrete squares, and a finite control that moves back and forth on the tape, reading and writing symbols. For a formal definition, see, e.g., Sipser [228] or Cook [72].

In this survey, I won't define Turing machines, for the simple reason that *if you know any programming language—C, Java, Python, etc.—then you already know something that's equivalent to Turing machines for our purposes.* More precisely, the *Church-Turing Thesis* holds that virtually any model of computation one can define will be equivalent to Turing machines, in the sense that Turing machines can simulate that model and vice versa. A modern refinement, the *Extended*

Turing machines $M_1, M_2, \ldots$ (that is, for all t, runs $M_1, \ldots, M_t$ for t steps each), halting when and if any M_i outputs a valid solution to one's NP search problem. If we know $\mathsf{P} = \mathsf{NP}$, then we know this particular algorithm will find a valid solution, whenever one exists, in polynomial time—because clearly *some* M_i does so, and all the machines other than M_i increase the total running time by "only" a polynomial factor! With more work, one can even decrease this to a constant factor. Admittedly, however, the polynomial or constant factor will be so enormous as to negate this algorithm's practical use.

Church-Turing Thesis, says that moreover, these simulations will incur at most a polynomial overhead in time and memory. Nowadays, most computer scientists and physicists conjecture that *quantum computation* provides a counterexample to the Extended Church-Turing Thesis—possibly the only counterexample that can be physically realized. In Section 5.5, I'll say a bit about how this changes the story. It's also conceivable that access to a true random-number generator would let us violate the Extended Church-Turing Thesis, although most computer scientists conjecture that it doesn't, for reasons that I'll explain in Section 5.4.1. On the other hand, as long as we're talking only about *classical, digital, deterministic* computation, the Extended Church-Turing Thesis remains on extremely solid ground.

If we accept this, then there's a well-defined notion of "solvable in polynomial time," which is independent of the low-level details of the computer's architecture: the instruction set, the specific rules for accessing memory, etc. This licenses us to ignore those details. The main caveat here is that there must be no *a-priori* limit on how much memory the computer can address, even though any program that runs for finite time will only address a finite amount of memory.[5,6]

We can now define P and NP, in terms of Turing machines for concreteness—but, because of the Extended Church-Turing Thesis, the reader is free to substitute other computing formalisms such as Lisp programs, λ-calculus, stylized assembly language, or cellular automata.

A *language*—the term is historical, coming from when theoretical computer science was closely connected to linguistics—just means a set of binary strings, $L \subseteq \{0,1\}^*$, where $\{0,1\}^*$ is the set of all binary strings of all (finite) lengths. Of course a language can be infinite, even though every string in the language is finite. One example is the language consisting of all palindromes: for instance, 00, 11, 0110, 11011, etc., but not 001 or 1100. A more interesting example is the language consisting of all binary encodings of prime numbers: for instance, 10, 11, 101, and 111, but not 100.

A binary string $x \in \{0,1\}^*$, for which we want to know whether $x \in L$, is called an *instance* of the general problem of deciding membership in L. Given a Turing machine M and an instance x, we let $M(x)$ denote M run on input x (say, on a tape initialized to $\cdots 0\#x\#0\cdots$, or x surrounded by delimiters and blank or 0 symbols). We say that $M(x)$ *accepts* if it eventually halts and enters an "accept" state, and we say that M *decides* the language L if for all $x \in \{0,1\}^*$,

$$x \in L \iff M(x) \text{ accepts.}$$

The machine M may also contain a "reject" state, which M enters to signify that it has halted without accepting. Let $|x|$ be the length of x (i.e., the number of bits). Then we say M is *polynomial-time* if there exists a polynomial p such that $M(x)$ halts, either accepting or rejecting, after at most $p(|x|)$ steps, for all $x \in \{0,1\}^*$.

Now P, or Polynomial-Time, is the class of all languages L for which there exists a Turing machine M that decides L in polynomial time. Also, NP, or Nondeterministic Polynomial-Time,

[5]The reason for this caveat is that, if a programming language were inherently limited to (say) 64K of memory, there would be only finitely many possible program behaviors, so in principle we could just cache everything in a giant lookup table. Many programming languages do impose a finite upper bound on the addressable memory, but they could easily be generalized to remove this restriction (or one could consider programs that store information on external I/O devices).

[6]I should stress that, once we specify which computational models we have in mind—Turing machines, Intel machine code, etc.—the polynomial-time equivalence of those models is typically a *theorem*, though a rather tedious one. The "thesis" of the Extended Church-Turing Thesis, the part not susceptible to proof, is that all *other* "reasonable" models of digital computation will also be equivalent to those models.

is the class of languages L for which there exists a polynomial-time Turing machine M, and a polynomial p, such that for all $x \in \{0,1\}^*$,

$$x \in L \iff \exists w \in \{0,1\}^{p(|x|)} \ M(x,w) \text{ accepts.}$$

In other words, NP is the class of languages L for which, whenever $x \in L$, there exists a polynomial-size "witness string" w, which enables a polynomial-time "verifier" M to recognize that indeed $x \in L$. Conversely, whenever $x \notin L$, there must be no w that causes $M(x,w)$ to accept.

There's an earlier definition of NP, which explains its ungainly name. Namely, we can define a *nondeterministic Turing machine* as a Turing machine that "when it sees a fork in the road, takes it": that is, that's allowed to transition from a single state at time t to multiple possible states at time $t+1$. We say that a machine "accepts" its input x, if there *exists* a list of valid transitions between states, $s_1 \to s_2 \to s_3 \to \cdots$, that the machine could make on input x that terminates in an accepting state s_{Accept}. The machine "rejects" if there's no such accepting path. The "running time" of such a machine is the maximum number of steps taken along *any* path, until the machine either accepts or rejects. We can then define NP as the class of all languages L for which there exists a nondeterministic Turing machine that decides L in polynomial time. It's clear that NP, so defined, is equivalent to the more intuitive verifier definition that we gave earlier. In one direction, if we have a polynomial-time verifier M, then a nondeterministic Turing machine can create paths corresponding to all possible witness strings w, and accept if and only if there exists a w such that $M(x,w)$ accepts. In the other direction, if we have a nondeterministic Turing machine M', then a verifier can take as its witness string w a description of a claimed path that causes $M'(x)$ to accept, then check that the path indeed does so.

Clearly $\mathsf{P} \subseteq \mathsf{NP}$, since an NP verifier M can just ignore its witness w, and try to decide in polynomial time whether $x \in L$ itself. The central conjecture is that this containment is strict.

Conjecture 1 $\mathsf{P} \neq \mathsf{NP}$.

2.1 NP-Completeness

A further concept, not part of the statement of $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ but central to any discussion of it, is NP-*completeness*. To explain this requires a few more definitions. An *oracle Turing machine* is a Turing machine that, at any time, can submit an instance x to an "oracle": a device that, in a single time step, returns a bit indicating whether x belongs to some given language L. Though it sounds fanciful, this notion is what lets us relate different computational problems to each other, and as such is one of the central concepts in computer science. An oracle that answers all queries consistently with L is called an L-*oracle*, and we write M^L to denote the (oracle) Turing machine M with L-oracle. We can then define P^L, or P *relative to* L, as the class of all languages L' for which there exists an oracle machine M such that M^L decides L' in polynomial time. If $L' \in \mathsf{P}^L$, then we also write $L' \leq_{\mathsf{P}}^T L$, which means "L' is polynomial-time Turing-reducible to L." Note that polynomial-time Turing-reducibility is indeed a partial order relation (i.e., it's transitive and reflexive).

A language L is NP-*hard* (technically, NP-hard under Turing reductions[7]) if $\mathsf{NP} \subseteq \mathsf{P}^L$. Informally, NP-hard means "at least as hard as any NP problem, under partial ordering by reductions."

[7]In practice, often one only needs a special kind of Turing reduction called a *many-one reduction* or *Karp reduction*, which is a polynomial-time algorithm that maps every yes-instance of L' to a yes-instance of L, and every no-instance of L' to a no-instance of L. The additional power of Turing reductions—to make multiple queries to the L-oracle (with

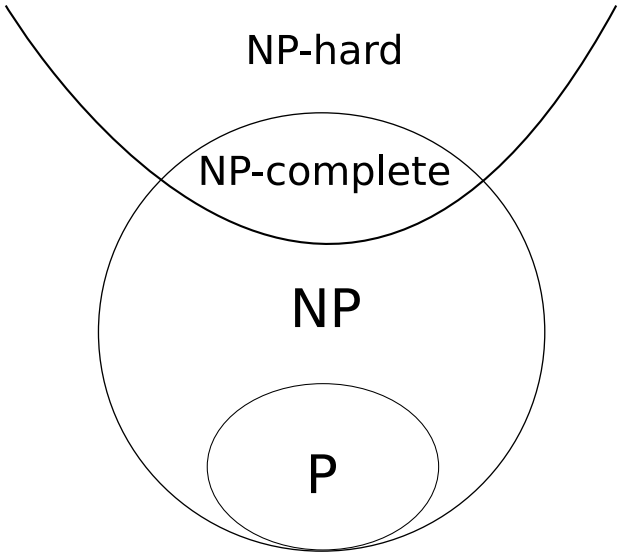

Figure 1: P, NP, NP-hard, and NP-complete

That is, if we had a black box for an NP-hard problem, we could use it to solve all NP problems in polynomial time. Also, L is NP-*complete* if L is NP-hard *and* $L \in \mathsf{NP}$. Informally, NP-complete problems are the hardest problems in NP, in the sense that an efficient algorithm for any of them would yield efficient algorithms for all NP problems. (See Figure 2.1.)

A priori, it's not completely obvious that NP-hard or NP-complete problems even exist. The great discovery of theoretical computer science in the 1970s was that hundreds of problems of practical importance fall into these classes—giving order to what had previously looked like a random assortment of incomparable hard problems. Indeed, among the real-world NP problems that aren't known to be in P, the great majority (though not all of them) are known to be NP-complete.

More concretely, consider the following languages:

- 3SAT is the language consisting of all Boolean formulas φ over n variables, which consist of ANDs of "3-clauses" (i.e., ORs of up to 3 variables or their negations), such that there exists at least one assignment that satisfies φ. (Or strictly speaking, all *encodings* of such formulas as binary strings, under some fixed encoding scheme whose details don't normally matter.) Here's an example, for which one can check that there's *no* satisfying assignment:

  $$(x \vee y \vee z) \wedge (\overline{x} \vee \overline{y} \vee \overline{z}) \wedge (x \vee \overline{y}) \wedge (\overline{x} \vee y) \wedge (y \vee \overline{z}) \wedge (\overline{y} \vee z)$$

  This translates to: at least one of x, y, z is true, at least one of x, y, z is false, $x = y$, and $y = z$.

- HAMILTONCYCLE is the language consisting of all undirected graphs, for which there exists a cycle that visits each vertex exactly once: that is, a Hamilton cycle. (Again, here we mean

later queries depending on the outcomes of earlier ones), post-process the results of those queries, etc.—is needed only in a minority of cases. For that reason, most sources define NP-hardness in terms of many-one reductions. Nevertheless, for conceptual simplicity, throughout this survey I'll talk in terms of Turing reductions.

all *encodings* of graphs as bit strings, under some fixed encoding scheme. For a string to belong to HamiltonCycle, it must be a valid encoding of some graph, *and* that graph must contain a Hamilton cycle.)

- TSP (Traveling Salesperson Problem) is the language consisting of all encodings of ordered pairs $\langle G, k \rangle$, such that G is a graph with positive integer weights, k is a positive integer, and G has a Hamilton cycle of total weight at most k.
- Clique is the language consisting of all encodings of ordered pairs $\langle G, k \rangle$, such that G is an undirected graph, k is a positive integer, and G contains a clique (i.e., a subset of vertices all connected to each other) with at least k vertices.
- SubsetSum is the language consisting of all encodings of positive integer tuples $\langle a_1, \ldots, a_k, b \rangle$, for which there exists a subset of the a_i's that sums to b.
- 3Col is the language consisting of all encodings of undirected graphs G that are 3-*colorable* (that is, the vertices of G can be colored red, green, or blue, so that no two adjacent vertices are colored the same).

All of these languages are easily seen to be in NP: for example, 3Sat is in NP because we can simply give a satisfying assignment to φ as a yes-witness, HamiltonCycle is in NP because we can give the Hamilton cycle, etc. The famous *Cook-Levin Theorem* says that one of these problems—3Sat—is also NP-hard, and hence NP-complete.

Theorem 2 (Cook-Levin Theorem [73, 157]) 3Sat *is* NP*-complete.*

A proof of Theorem 2 can be found in any theory of computing textbook (for example, [228]). Here I'll confine myself to saying that Theorem 2 can be proved in three steps, each of them routine from today's standpoint:

(1) One constructs an artificial language that's "NP-complete essentially by definition": for example,

$$L = \left\{ \left(\langle M \rangle, x, 0^s, 0^t \right) : \exists w \in \{0,1\}^s \text{ such that } M(x,w) \text{ accepts in } \leq t \text{ steps} \right\},$$

where $\langle M \rangle$ is a description of the Turing machine M. (Here s and t are encoded in so-called "unary notation," to prevent a polynomial-size input from corresponding to an exponentially-large witness string or exponential amount of time, and thereby keep L in NP.)

(2) One then reduces L to the CircuitSat problem, where we're given as input a description of a Boolean circuit C built of AND, OR, and NOT gates, and asked whether there exists an assignment $x \in \{0,1\}^n$ for the input bits such that $C(x) = 1$. To give this reduction, in turn, is more like electrical engineering than mathematics: given a Turing machine M, one simply builds up a Boolean logic circuit that simulates the action of M on the input (x, w) for t time steps, whose size is polynomial in the parameters $|\langle M \rangle|$, $|x|$, s, and t, and which outputs 1 if and only if M ever enters its accept state.

(3) Finally, one reduces CircuitSat to 3Sat, by creating a new variable for each gate in the Boolean circuit C, and then creating clauses to enforce that the variable for each gate G equals the AND, OR, or NOT (as appropriate) of the variables for G's inputs. For example, one can express the constraint $a \wedge b = c$ by

$$(a \vee \overline{c}) \wedge (b \vee \overline{c}) \wedge \left(\overline{a} \vee \overline{b} \vee c\right).$$

One then constrains the variable for the final output gate to be 1, yielding a 3Sat instance φ that is satisfiable if and only if the CircuitSat instance was (i.e., iff there existed an x such that $C(x) = 1$).

Note that the algorithms to reduce L to CircuitSat and CircuitSat to 3Sat—i.e., to convert M to C and C to φ—run in polynomial time (actually linear time), so we indeed preserve NP-hardness. Also, the reason for the 3 in 3Sat is simply that an AND or OR gate has one output bit and two input bits, so it relates three bits in total. The analogous 2Sat problem turns out to be in P.

Once one knows that 3Sat is NP-complete, "the floodgates are open." One can then prove that countless other NP problems are NP-complete by reducing 3Sat to them, and then reducing those problems to others, and so on. The first indication of how pervasive NP-completeness really was came from Karp [134] in 1972. He showed, among many other results:

Theorem 3 (Karp [134]) HamiltonCycle, TSP, Clique, SubsetSum, *and* 3Col *are all* NP*-complete.*

Today, so many combinatorial search problems have been proven NP-complete that, whenever one encounters a new such problem, a useful rule of thumb is that it's "NP-complete unless it has a good reason not to be!"

Note that, if any NP-complete problem is in P, then all of them are, and P = NP (since every NP problem can first be reduced to the NP-complete one, and then solved in polynomial time). Conversely, if any NP-complete problem is not in P, then none of them are, and P ≠ NP.

One application of NP-completeness is to reduce the number of logical quantifiers needed to state the P ≠ NP conjecture, and thereby make it intuitively easier to grasp. A Σ_k*-sentence* is a sentence with k quantifiers over integers (or objects that can be encoded as integers), beginning with an existential ($\exists$) quantifier; a Π_k*-sentence* also has k quantifiers, but begins with a universal ($\forall$) quantifier. Let $\mathcal{PT}$ be the set of all polynomial-time Turing machines, and given a language L, let $L(x)$ be the *characteristic function* of L: that is, $L(x) = 1$ if $x \in L$ and $L(x) = 0$ otherwise. Then a "naïve" statement of P ≠ NP would be as a Σ_3-sentence:

$$\exists L \in \mathsf{NP}\ \forall M \in \mathcal{PT}\ \exists x\ M(x) \neq L(x).$$

(Here, by quantifying over all languages in NP, we really mean quantifying over all verification algorithms that define such languages.) But once we know that 3Sat (for example) is NP-complete, we can state P ≠ NP as just a Π_2-sentence:

$$\forall M \in \mathcal{PT}\ \exists x\ M(x) \neq 3\textsc{Sat}(x).$$

In words, we can pick any NP-complete problem we like; then P ≠ NP is equivalent to the statement that *that* problem is not in P.

2.2 Other Core Concepts

A few more concepts give a fuller picture of the $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ question, and will be referred to later in the survey. In this section, I'll restrict myself to concepts that were explored in the 1970s, around the same time as $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ itself was formulated, and that are covered alongside $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ in undergraduate textbooks. Other important concepts, such as nonuniformity, randomness, and one-way functions, will be explained as needed in Section 5.

2.2.1 Search, Decision, and Optimization

For technical convenience, P and NP are defined in terms of languages or "decision problems," which have a single yes-or-no bit as the desired output (i.e., given an input x, is $x \in L$?). To put practical problems into this decision format, typically we ask something like: *does there exist* a solution that satisfies the following list of constraints? But of course, in real life we don't merely want to know whether a solution exists; we want to *find* a solution whenever there is one! And given the many examples in mathematics where explicitly finding an object is harder than proving its existence, one might worry that this would also occur here. Fortunately, though, shifting our focus from decision problems to search problems doesn't change the $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ question at all, because of the following classic observation.

Proposition 4 *If $\mathsf{P} = \mathsf{NP}$, then for every language $L \in \mathsf{NP}$ (defined by a verifier M), there's a polynomial-time algorithm that actually finds a witness $w \in \{0,1\}^{p(n)}$ such that $M(x,w)$ accepts, for all $x \in L$.*

Proof. The idea is to learn the bits of an accepting witness $w = w_1 \cdots w_{p(n)}$ one by one, by asking a series of NP decision questions. For example:

- Does there exist a w such that $M(x,w)$ accepts and $w_1 = 0$?

If the answer is "yes," then next ask:

- Does there exist a w such that $M(x,w)$ accepts, $w_1 = 0$, and $w_2 = 0$?

Otherwise, next ask:

- Does there exist a w such that $M(x,w)$ accepts, $w_1 = 1$, and $w_2 = 0$?

Continue in this manner until all $p(n)$ bits of w have been set. (This can also be seen as a binary search on the set of $2^{p(n)}$ possible witnesses.) ■

Note that there *are* problems for which finding a solution is believed to be much harder than deciding whether one exists. A classic example, as it happens, is the problem of finding a Nash equilibrium of a matrix game. Here Nash's theorem guarantees that an equilibrium always exists, but an important 2006 result of Daskalakis et al. [78] gave evidence that there's no polynomial-time algorithm to *find* an equilibrium.[8] The upshot of Proposition 4 is just that search and decision are equivalent for the NP-*complete* problems.

[8] Technically, Daskalakis et al. showed that the search problem of finding a Nash equilibrium is complete for a complexity class called PPAD. This could be loosely interpreted as saying that the problem is "as close to NP-hard as it could possibly be, subject to Nash's theorem showing why the decision version is trivial."

In practice, perhaps even more common than search problems are *optimization problems*, where we have some efficiently-computable cost function, say $C : \{0,1\}^n \rightarrow \{0,1,\ldots,2^{p(n)}\}$, and the goal is to find a solution $x \in \{0,1\}^n$ that minimizes $C(x)$. Fortunately, we can always reduce optimization problems to search and decision problems, by simply asking to find a solution x such that $C(x) \leq K$, and doing a binary search to find the smallest K for which such an x still exists. So again, if $\mathsf{P} = \mathsf{NP}$ then all NP optimization problems are solvable in polynomial time. On the other hand, it's important to remember that, while "is there an x such that $C(x) \leq K$?" is an NP question, "does $\min_x C(x) = K$?" and "does x^* minimize $C(x)$?" are presumably *not* NP questions in general, because no single x is a witness to a yes-answer.

More generally, the fact that decision, search, and optimization all hinge on the same $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ question has meant that many people, including experts, freely abuse language by referring to search and optimization problems as "NP-complete." Strictly they should call such problems NP-hard—we defined NP-hardness for languages, but the concept can be generalized to search and optimization problems—while reserving "NP-complete" for suitable associated decision problems.

2.2.2 The Twilight Zone: Between P and NP-complete

We say a language L is NP-*intermediate* if $L \in \mathsf{NP}$, but L is neither in P nor NP-complete. Based on experience, one might hope not only that $\mathsf{P} \neq \mathsf{NP}$, but that there'd be a *dichotomy*, with all NP problems either in P or else NP-complete. However, a classic result by Ladner [149] rules that possibility out.

Theorem 5 (Ladner [149]) *If $\mathsf{P} \neq \mathsf{NP}$, then there exist NP-intermediate languages.*

While Theorem 5 is theoretically important, the NP-intermediate problems that it yields are extremely artificial (requiring diagonalization to construct). On the other hand, as we'll see, there are also problems of real-world importance—particularly in cryptography, algebra, and number theory—that are believed to be NP-intermediate, and a proof of $\mathsf{P} \neq \mathsf{NP}$ could leave the status of those problems open. (Of course, a proof of $\mathsf{P} = \mathsf{NP}$ would mean there were *no* NP-intermediate problems, since every NP problem would then be both NP-complete and in P.)

2.2.3 coNP and the Polynomial Hierarchy

Let $\overline{L} = \{0,1\}^* \setminus L$ be the *complement* of L: that is, the set of strings not in L. Then the complexity class

$$\mathsf{coNP} := \left\{\overline{L} : L \in \mathsf{NP}\right\}$$

consists of the complements of all languages in NP. Note that this is *not* the same as $\overline{\mathsf{NP}}$, the set of all non-NP languages! Rather, $L \in \mathsf{coNP}$ means that whenever $x \notin L$, there's a short proof of non-membership that can be efficiently verified. If $L \in \mathsf{NP}$, then $\overline{L} \in \mathsf{coNP}$ and vice versa. Likewise, if L is NP-complete, then $\overline{L}$ is coNP-complete (that is, in coNP and NP-hard) and vice versa. So for example, along with the NP-complete satisfiability we have the coNP-complete *un*satisfiability, along with the NP-complete HamiltonCycle we have the coNP-complete $\overline{\text{HamiltonCycle}}$ (consisting of all encodings of graphs that *lack* a Hamilton cycle), etc.

A natural question is whether NP is *closed under complement*: that is, whether $\mathsf{NP} = \mathsf{coNP}$. If $\mathsf{P} = \mathsf{NP}$, then certainly $\mathsf{P} = \mathsf{coNP}$, and hence $\mathsf{NP} = \mathsf{coNP}$ also. On the other hand, we could imagine a world where $\mathsf{NP} = \mathsf{coNP}$ even though $\mathsf{P} \neq \mathsf{NP}$. In that world, there would always be

short proofs of *un*satisfiability (or of the *non*existence of cliques, Hamilton cycles, etc.), but those proofs could be intractable to find. A generalization of the $\mathsf{P} \neq \mathsf{NP}$ conjecture says that this doesn't happen:

Conjecture 6 $\mathsf{NP} \neq \mathsf{coNP}$.

A further generalization of P, NP, and coNP is the *polynomial hierarchy* PH. Defined by analogy with the *arithmetic hierarchy* in computability theory, PH is an infinite sequence of classes whose zeroth level equals P, and whose k^{th} level (for $k \geq 1$) consists of all problems that are in P^L or NP^L or coNP^L, for some language L in the $(k-1)^{st}$ level. More succinctly, we write $\Sigma_0^{\mathsf{P}} = \mathsf{P}$, and

$$\Delta_k^{\mathsf{P}} = \mathsf{P}^{\Sigma_{k-1}^{\mathsf{P}}}, \qquad \Sigma_k^{\mathsf{P}} = \mathsf{NP}^{\Sigma_{k-1}^{\mathsf{P}}}, \qquad \Pi_k^{\mathsf{P}} = \mathsf{coNP}^{\Sigma_{k-1}^{\mathsf{P}}}$$

for all $k \geq 1$.[9] A more intuitive definition of PH is as the class of languages that are definable using a polynomial-time predicate with a constant number of alternating universal and existential quantifiers: for example, $L \in \Pi_2^{\mathsf{P}}$ if and only if there exists a polynomial-time machine M and polynomial p such that for all x,

$$x \in L \iff \forall w \in \{0,1\}^{p(|x|)} \, \exists z \in \{0,1\}^{p(|x|)} \;\; M(x,w,z) \text{ accepts.}$$

NP is then the special case with just one existential quantifier, over witness strings w.

If $\mathsf{P} = \mathsf{NP}$, then the entire PH "recursively unwinds" down to P: for example,

$$\Sigma_2^{\mathsf{P}} = \mathsf{NP}^{\mathsf{NP}} = \mathsf{NP}^{\mathsf{P}} = \mathsf{NP} = \mathsf{P}.$$

Moreover, one can show that if $\Sigma_k^{\mathsf{P}} = \Pi_k^{\mathsf{P}}$ or $\Sigma_k^{\mathsf{P}} = \Sigma_{k+1}^{\mathsf{P}}$ for any k, then all the levels above the k^{th} come "crashing down" to $\Sigma_k^{\mathsf{P}} = \Pi_k^{\mathsf{P}}$. So for example, if $\mathsf{NP} = \mathsf{coNP}$, then $\mathsf{PH} = \mathsf{NP} = \mathsf{coNP}$ as well. On the other hand, a collapse at the k^{th} level isn't known to imply a collapse at any *lower* level. Thus, we get an infinite sequence of stronger and stronger conjectures: first $\mathsf{P} \neq \mathsf{NP}$, then $\mathsf{NP} \neq \mathsf{coNP}$, then $\Sigma_2^{\mathsf{P}} \neq \Pi_2^{\mathsf{P}}$, and so on. In the limit, we can conjecture the following:

Conjecture 7 *All the levels of* PH *are distinct—i.e., the infinite hierarchy is strict.*

This is a generalization of $\mathsf{P} \neq \mathsf{NP}$ that many computer scientists believe—an intuition being that it would seem strange for the hierarchy to collapse at, say, the 17^{th} level, without collapsing all the way down to P or NP. Conjecture 7 has many important consequences that aren't known to follow from $\mathsf{P} \neq \mathsf{NP}$ itself.

It's also interesting to consider $\mathsf{NP} \cap \mathsf{coNP}$, which is the class of languages that admit short, easily-checkable proofs for both membership *and* non-membership. Here's yet another strengthening of the $\mathsf{P} \neq \mathsf{NP}$ conjecture:

Conjecture 8 $\mathsf{P} \neq \mathsf{NP} \cap \mathsf{coNP}$.

Of course, if $\mathsf{NP} = \mathsf{coNP}$, then the $\mathsf{P} \stackrel{?}{=} \mathsf{NP} \cap \mathsf{coNP}$ question becomes equivalent to the original $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ question. But it's conceivable that $\mathsf{P} = \mathsf{NP} \cap \mathsf{coNP}$ even if $\mathsf{NP} \neq \mathsf{coNP}$.

[9] In defining the k^{th} level of the hierarchy, we could also have given oracles for Π_{k-1}^{P} rather than $\Sigma_{k-1}^{\mathsf{P}}$: it doesn't matter. Note also that "an oracle for complexity class $\mathcal{C}$" should be read as "an oracle for any $\mathcal{C}$-complete language L."

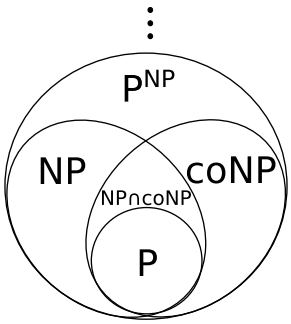

Figure 2: The polynomial hierarchy

2.2.4 Factoring and Graph Isomorphism

As an application of these concepts, let's consider two languages that are suspected to be NP-intermediate. First, FAC—a language variant of the factoring problem—consists of all ordered pairs of positive integers $\langle N, k \rangle$ such that N has a nontrivial divisor at most k. Clearly a polynomial-time algorithm for FAC can be converted into a polynomial-time algorithm to output the prime factorization (by repeatedly doing binary search to peel off N's smallest divisor), and vice versa. Second, GRAPHISO—that is, graph isomorphism—consists of all encodings of pairs of undirected graphs $\langle G, H \rangle$, such that $G \cong H$. It's easy to see to see that FAC and GRAPHISO are both in NP.

More interestingly, FAC is actually in $\mathsf{NP} \cap \mathsf{coNP}$. For one can prove that $\langle N, k \rangle \notin$FAC by exhibiting the unique prime factorization of N, and showing that it only involves primes greater than k.[10] But this has the striking consequence that *factoring can't be* NP*-complete unless* $\mathsf{NP} = \mathsf{coNP}$. The reason is the following.

Proposition 9 *If any* $\mathsf{NP} \cap \mathsf{coNP}$ *language is* NP*-complete, then* $\mathsf{NP} = \mathsf{coNP}$*, and hence* PH *collapses to* NP*.*

Proof. Suppose $L \in \mathsf{NP} \cap \mathsf{coNP}$. Then $\mathsf{P}^L \subseteq \mathsf{NP} \cap \mathsf{coNP}$, since one can prove the validity of every answer to every query to the L-oracle (whether the answer is 'yes' or 'no'). So if $\mathsf{NP} \subseteq \mathsf{P}^L$, then $\mathsf{NP} \subseteq \mathsf{NP} \cap \mathsf{coNP}$ and hence $\mathsf{NP} = \mathsf{coNP}$. ■

GRAPHISO is not quite known to be in $\mathsf{NP} \cap \mathsf{coNP}$. However, it's been proven to be in $\mathsf{NP} \cap \mathsf{coNP}$ under a plausible assumption about pseudorandom generators [143]—and even with no assumptions, Boppana, Håstad, Zachos [53] proved the following.

Theorem 10 ([53]) *If* GRAPHISO *is* NP*-complete, then* PH *collapses to* Σ_2^{P}*.*

As this survey was being written, Babai [33] announced the following breakthrough result.

Theorem 11 (Babai [33]) GRAPHISO *is solvable in subexponential time: specifically* $2^{2^{\widetilde{O}(\sqrt{\log n})}}$*, where the* $\widetilde{O}$ *hides a factor polynomial in* $\log \log n$*.*[11]

[10] This requires one nontrivial result, that every prime number has a succinct certificate—or in other words, that primality testing is in NP [198]. Since 2002, it is even known that primality testing is in P [15].

[11] Originally, Babai had claimed a GRAPHISO algorithm running in *quasipolynomial* time ($n^{k(\log n)^c}$). On January 4, 2017, however, he posted an announcement scaling back the running time claim to subexponential.

The best previously-known bound, due to Babai and Luks [34] from 1983, had been $2^{O(\sqrt{n \log n})}$. Of course, Theorem 11 gives even more dramatic evidence that GraphIso is not NP-complete: if it was, then *all* NP problems would be solvable in subexponential time as well. Based on experience—namely, that known algorithms can solve GraphIso extremely quickly for almost any graphs we can generate in practice—many computer scientists have conjectured for decades that GraphIso is in P, and certainly Theorem 11 is consistent with that conviction, but there remain significant obstacles to proving it.[12]

2.2.5 Space Complexity

PSPACE is the class of languages L decidable by a Turing machine that uses a polynomial number of bits of *space* or *memory*, with no restriction on the number of time steps. Certainly $\mathsf{P} \subseteq \mathsf{PSPACE}$, since in t time steps, a serial algorithm can access at most t memory cells. More generally, it's not hard to see that $\mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PH} \subseteq \mathsf{PSPACE}$, since in an expression like $\forall x \exists y \, \varphi(x, y)$, a PSPACE machine can loop over all possible values for x and y, using exponential time but reusing the same memory for each x, y pair. However, *none* of these containments have been proved to be strict.

The following conjecture—asserting that polynomial space is strictly stronger than polynomial time—is perhaps second only to $\mathsf{P} \neq \mathsf{NP}$ itself in notoriety.

Conjecture 12 $\mathsf{P} \neq \mathsf{PSPACE}$.

If $\mathsf{P} \neq \mathsf{NP}$, then certainly $\mathsf{P} \neq \mathsf{PSPACE}$ as well, but the converse isn't known.

One can also define a nondeterministic variant of PSPACE, called NPSPACE. But a 1970 result called *Savitch's Theorem* [217] shows that actually $\mathsf{PSPACE} = \mathsf{NPSPACE}$.[13] The reasons for this are extremely specific to space, and don't seem to suggest any avenue to proving $\mathsf{P} = \mathsf{NP}$, the analogous statement for time.

2.2.6 Counting Complexity

Given an NP search problem, besides asking whether a solution exists, it's also natural to ask how many solutions there are. To capture this, in 1979 Valiant [247] defined the class #P (pronounced "sharp-P", not "hashtag-P"!) of combinatorial counting problems. Formally, a function $f : \{0,1\}^* \rightarrow \mathbb{N}$ is in #P if and only if there's a polynomial-time Turing machine M, and a polynomial p, such that for all $x \in \{0,1\}^*$,

$$f(x) = \left| \left\{ w \in \{0,1\}^{p(|x|)} : M(x, w) \text{ accepts} \right\} \right|.$$

Note that, unlike P, NP, and so on, #P is not a class of languages (i.e., decision problems). However, there are two ways we can compare #P to language classes.

The first is by considering $\mathsf{P}^{\#\mathsf{P}}$: that is, P with a #P oracle. We then have $\mathsf{NP} \subseteq \mathsf{P}^{\#\mathsf{P}} \subseteq \mathsf{PSPACE}$, as well as the following highly non-obvious inclusion, called *Toda's Theorem*.

[12] In particular, *group isomorphism*—that is, the problem of deciding whether two groups of order n, given by their multiplication tables, are isomorphic—is clearly solvable in $\sim n^{\log n}$ time and clearly reducible to GraphIso, but no algorithm for it better than $\sim n^{\log n}$ is known.

[13] A further surprising result from 1987, called the *Immerman-Szelepcsényi Theorem* [121, 239], says that the class of languages decidable by a nondeterministic machine in $f(n)$ space is closed under complement, for every "reasonable" memory bound $f(n)$. (By contrast, Savitch's Theorem produces a quadratic blowup when simulating nondeterministic space by deterministic space, and it remains open whether that blowup can be removed.) This further illustrates how space complexity behaves differently than we expect time complexity to behave.

Theorem 13 (Toda [243]) $\mathsf{PH} \subseteq \mathsf{P}^{\#\mathsf{P}}$.

The second way is by considering a complexity class called PP (Probabilistic Polynomial-Time). PP can be defined as the class of languages $L \subseteq \{0,1\}^*$ for which there exist #P functions f and g such that for all inputs $x \in \{0,1\}^*$,

$$x \in L \iff f(x) \geq g(x).$$

Equivalently, PP is the class of languages L for which there exists a probabilistic polynomial-time algorithm that merely needs to guess whether $x \in L$, for each input x, with *some* probability greater than $1/2$. It's not hard to see that $\mathsf{NP} \subseteq \mathsf{PP} \subseteq \mathsf{P}^{\#\mathsf{P}}$. More interestingly, one can use binary search to show that $\mathsf{P}^{\mathsf{PP}} = \mathsf{P}^{\#\mathsf{P}}$, so in that sense PP is "almost as strong as #P."

In practice, for any known NP-complete problem (3Sat, Clique, SubsetSum, etc.), the counting version of that problem (denoted #3Sat, #Clique, #SubsetSum, etc.) is #P-complete. Indeed, it's open whether there's any NP-complete problem that violates this rule. However, the converse statement is false. For example, the problem of deciding whether a graph has a perfect matching—that is, a set of edges that touches each vertex exactly once—is in P, but Valiant [247] showed that counting the *number* of perfect matchings is #P-complete.

The #P-complete problems are believed to be "genuinely much harder" than the NP-complete problems, in the sense that—in contrast to the situation with PH—even if $\mathsf{P} = \mathsf{NP}$ we'd still have no idea how to prove $\mathsf{P} = \mathsf{P}^{\#\mathsf{P}}$. On the other hand, we do have the following nontrivial result.

Theorem 14 (Stockmeyer [232]) *Suppose* $\mathsf{P} = \mathsf{NP}$. *Then in polynomial time, we could approximate any* #P *function to within a factor of* $1 \pm \frac{1}{p(n)}$, *for any polynomial* p.

2.2.7 Beyond Polynomial Resources

Of course, we can consider many other time and space bounds besides polynomial. Before entering into this, I should offer a brief digression on the use of asymptotic notation in theoretical computer science, since such notation will also be used later in the survey.

- $f(n)$ is $O(g(n))$ if there exist nonnegative constants A, B such that $f(n) \leq Ag(n) + B$ for all n (i.e., g is an asymptotic upper bound on f).
- $f(n)$ is $\Omega(g(n))$ if $g(n)$ is $O(f(n))$ (i.e., g is an asymptotic *lower* bound on f).
- $f(n)$ is $\Theta(g(n))$ if $f(n)$ is $O(g(n))$ and $g(n)$ is $O(f(n))$ (i.e., f and g grow at the same asymptotic rate).
- $f(n)$ is $o(g(n))$ if for all positive A, there exists a B such that $f(n) \leq Ag(n) + B$ for all n (i.e., g is a *strict* asymptotic upper bound on f).[14]

Now let $\mathsf{TIME}(f(n))$ be the class of languages decidable in $O(f(n))$ time, let $\mathsf{NTIME}(f(n))$ be the class decidable in nondeterministic $O(f(n))$ time—that is, with a witness of size $O(f(n))$

[14] If $f(n)$ is $O(g(n))$ but not $\Theta(g(n))$, that does *not* necessarily mean that $f(n)$ is $o(g(n))$: for example, consider $f(n) = 2n(n/2 - \lfloor n/2 \rfloor)$ and $g(n) = n$. Of course, such examples don't arise often in practice.

that's verified in $O(f(n))$ time—and let $\mathsf{SPACE}(f(n))$ be the class decidable in $O(f(n))$ space.[15] We can then write $\mathsf{P} = \bigcup_k \mathsf{TIME}\left(n^k\right)$ and $\mathsf{NP} = \bigcup_k \mathsf{NTIME}\left(n^k\right)$ and $\mathsf{PSPACE} = \bigcup_k \mathsf{SPACE}\left(n^k\right)$. It's also interesting to study the exponential versions of these classes:

$$\mathsf{EXP} = \bigcup_k \mathsf{TIME}\left(2^{n^k}\right),$$
$$\mathsf{NEXP} = \bigcup_k \mathsf{NTIME}\left(2^{n^k}\right),$$
$$\mathsf{EXPSPACE} = \bigcup_k \mathsf{SPACE}\left(2^{n^k}\right).$$

Note that by "exponential," here we mean not just $2^{O(n)}$, but $2^{p(n)}$ for any polynomial p.

Just like we have $\mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{NP}^{\mathsf{NP}} \subseteq \cdots \subseteq \mathsf{PSPACE}$, so we also have

$$\mathsf{EXP} \subseteq \mathsf{NEXP} \subseteq \mathsf{NEXP}^{\mathsf{NP}} \subseteq \cdots \subseteq \mathsf{EXPSPACE}.$$

Along with $\mathsf{P} \subseteq \mathsf{PSPACE}$ (Section 2.2.5), there's another fundamental relation between time and space classes:

Proposition 15 $\mathsf{PSPACE} \subseteq \mathsf{EXP}$.

Proof. Consider a deterministic machine whose state can be fully described by $p(n)$ bits of information (e.g., the contents of a polynomial-size Turing machine tape, plus a few extra bits for the location and internal state of tape head). Clearly such a machine has at most $2^{p(n)}$ possible states. Thus, after $2^{p(n)}$ steps, either the machine has halted, or else it's entered an infinite loop and will never accept. So to decide whether the machine accepts, it suffices to simulate it for $2^{p(n)}$ steps. ■

More generally, we get an infinite interleaved hierarchy of deterministic, nondeterministic, and space classes:

$$\mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PH} \subseteq \mathsf{PSPACE} \subseteq \mathsf{EXP} \subseteq \mathsf{NEXP} \subseteq \mathsf{EXPSPACE} \subseteq \cdots$$

There's also a "higher-up" variant of the $\mathsf{P} \neq \mathsf{NP}$ conjecture, which not surprisingly is *also* open:

Conjecture 16 $\mathsf{EXP} \neq \mathsf{NEXP}$.

We can at least prove a close relationship between the $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ and $\mathsf{EXP} \stackrel{?}{=} \mathsf{NEXP}$ questions, via a trick called *padding* or *upward translation*:

Proposition 17 *If* $\mathsf{P} = \mathsf{NP}$, *then* $\mathsf{EXP} = \mathsf{NEXP}$.

Proof. Let $L \in \mathsf{NEXP}$, and let its verifier run in $2^{p(n)}$ time for some polynomial p. Then consider the language

$$L' = \left\{ x0^{2^{p(|x|)}} : x \in L \right\},$$

[15] Unlike P or PSPACE, classes like $\mathsf{TIME}\left(n^2\right)$ and $\mathsf{SPACE}\left(n^3\right)$ can be sensitive to whether we're using Turing machines, RAM machines, or some other model of computation. Thus, I'll specify which I'm talking about whenever it's relevant.

which consists of the inputs in L, but "padded out with an exponential number of trailing zeroes." Then $L' \in \mathsf{NP}$, since verifying that $x \in \{0,1\}^n$ is in L takes $2^{p(n)}$ time, which is linear in $n + 2^{p(n)}$ (the length of $x0^{2^{p(|x|)}}$). So by assumption, $L' \in \mathsf{P}$ as well. But this means that $L \in \mathsf{EXP}$, since given $x \in \{0,1\}^n$ (an input for L), we can simply pad x out with $2^{p(n)}$ trailing zeroes ourselves, then run the algorithm for L that takes time polynomial in $n + 2^{p(n)}$. ■

For the same reason, if $\mathsf{P} = \mathsf{PSPACE}$, then $\mathsf{EXP} = \mathsf{EXPSPACE}$. On the other hand, padding only works in one direction: as far as anyone knows today, we could have $\mathsf{P} \neq \mathsf{NP}$ even if $\mathsf{EXP} = \mathsf{NEXP}$.

To summarize, $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ is just the tip of an iceberg; there seems to be an extremely rich structure both below and above the NP-complete problems. Until we can prove $\mathsf{P} \neq \mathsf{NP}$, however, most of that structure will remain conjectural.

3 Beliefs About $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$

Just as Hilbert's question turned out to have a negative answer, so too in this case, most computer scientists conjecture that $\mathsf{P} \neq \mathsf{NP}$: that there exist rapidly checkable problems that *aren't* rapidly solvable, and for which brute-force search is close to the best we can do. This is not a unanimous opinion. At least one famous computer scientist, Donald Knuth [144], has professed a belief that $\mathsf{P} = \mathsf{NP}$, while another, Richard Lipton [164], professes agnosticism. Also, in a poll of mathematicians and theoretical computer scientists conducted by William Gasarch [97] in 2002, there were 61 respondents who said $\mathsf{P} \neq \mathsf{NP}$, but also 9 who said $\mathsf{P} = \mathsf{NP}$. Admittedly, it can be hard to tell whether declarations that $\mathsf{P} = \mathsf{NP}$ are meant seriously, or are merely attempts to be contrarian. However, we can surely agree with Knuth and Lipton that we're far from understanding the limits of efficient computation, and that there are further surprises in store.

In this section, I'd like to explain why, *despite* our limited understanding, many of us feel roughly as confident about $\mathsf{P} \neq \mathsf{NP}$ as we do about (say) the Riemann Hypothesis, or other conjectures in math—not to mention empirical sciences—that most experts believe without proof.[16]

The first point is that, when we ask whether $\mathsf{P} = \mathsf{NP}$, we're not asking whether heuristic optimization methods (such as SAT-solvers) can *sometimes* do well in practice, or whether there are *sometimes* clever ways to avoid exponential search. If you believe, for example, that there's *any* cryptographic one-way function—that is, any transformation of inputs $x \to f(x)$ that's easy to compute but hard to invert (see Section 5.3.1)—then that's enough for $\mathsf{P} \neq \mathsf{NP}$. Such an f need not have any "nice" mathematical structure; it could simply be, say, the evolution function of some arbitrary cellular automaton.

It's sometimes claimed that, when we consider $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$, there's a "symmetry of ignorance": yes, we have no idea how to solve NP-complete problems in polynomial time, but we *also* have no idea how to prove that impossible, and therefore anyone's free to believe whatever they like. In my opinion, what breaks the symmetry is the *immense, well-known difficulty of proving lower bounds.* Simply put: even if we suppose $\mathsf{P} \neq \mathsf{NP}$, I don't believe there's any great mystery about why a proof has remained elusive. A rigorous impossibility proof is often a tall order, and many times in history—e.g., with Fermat's Last Theorem, the Kepler Conjecture, or the problem of squaring

[16] I like to joke that, if computer scientists had been physicists, we'd simply have declared $\mathsf{P} \neq \mathsf{NP}$ to be an observed law of Nature, analogous to the laws of thermodynamics. A Nobel Prize would even be given for the discovery of that law. (And in the unlikely event that someone later proved $\mathsf{P} = \mathsf{NP}$, a second Nobel Prize would be awarded for the law's overthrow.)

the circle—such a proof was requested *centuries* before mathematical understanding had advanced to the point where it became a realistic possibility. And as we'll see in Section 6, today we know something about the difficulty of proving even "baby" versions of $\mathsf{P} \neq \mathsf{NP}$; about the barriers that have been overcome and the others that remain.

By contrast, if $\mathsf{P} = \mathsf{NP}$, then there's at least a *puzzle* about why the entire software industry, over half a century, has failed to uncover any promising leads for, say, a fast algorithm to invert arbitrary one-way functions (just the algorithm itself, not necessarily a proof of correctness). The puzzle is heightened when we realize that, in many real-world cases—such as linear programming, primality testing, and network routing—fast methods to handle a problem in practice *did* come decades before a full theoretical understanding of why the methods worked.

Another reason to believe $\mathsf{P} \neq \mathsf{NP}$ comes from the hierarchy theorems, which we'll meet in Section 6.1. Roughly speaking, these theorems imply that "most" pairs of complexity classes are unequal; the trouble, in most cases, is merely that we can't prove this for *specific* pairs! For example, in the chain of complexity classes $\mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE} \subseteq \mathsf{EXP}$, we know that $\mathsf{P} \neq \mathsf{EXP}$, which implies that *at least one* of $\mathsf{P} \neq \mathsf{NP}$, $\mathsf{NP} \neq \mathsf{PSPACE}$, and $\mathsf{PSPACE} \neq \mathsf{EXP}$ must hold. So we might say: given the provable reality of a rich lattice of unequal complexity classes, one needs to offer a special argument if one thinks two classes collapse, but not necessarily if one thinks they're different.

To my mind, however, the strongest argument for $\mathsf{P} \neq \mathsf{NP}$ involves the thousands of problems that have been shown to be NP-complete, and the thousands of other problems that have been shown to be in P. If just one of these problems had turned out to be both NP-complete *and* in P, that would've immediately implied $\mathsf{P} = \mathsf{NP}$. Thus, we could argue, the $\mathsf{P} \neq \mathsf{NP}$ hypothesis has had thousands of chances to be "falsified by observation." Yet somehow, in every case, the NP-completeness reductions and the polynomial-time algorithms "miraculously" avoid meeting each other—a phenomenon that I once described as the "invisible fence" [7].

This phenomenon becomes particularly striking when we consider *approximation algorithms* for NP-hard problems, which return not necessarily an optimal solution but a solution within some factor of optimal. To illustrate, there's a simple polynomial-time algorithm that, given a 3SAT instance φ, finds an assignment that satisfies at least a $7/8$ fraction of the clauses.[17] Conversely, in 1997 Johan Håstad [115] proved the following striking result.

Theorem 18 (Håstad [115]) *Suppose there's a polynomial-time algorithm that, given as input a satisfiable* 3SAT *instance φ, outputs an assignment that satisfies at least a $7/8 + \varepsilon$ fraction of the clauses, where $\varepsilon > 0$ is any constant. Then $\mathsf{P} = \mathsf{NP}$.*

Theorem 18 is one (strong) version of the *PCP Theorem* [30, 31], which is considered one of the crowning achievements of theoretical computer science. The PCP Theorem yields many other examples of "sharp NP-completeness thresholds," where as we numerically adjust the required solution quality, an optimization problem undergoes a sudden "phase transition" from being in P to being NP-complete. Other times there's a gap between the region of parameter space known to

[17] Strictly speaking, this is for the variant of 3SAT in which every clause must have *exactly* 3 literals, rather than at most 3.

Also note that, if we allow the use of randomness, then we can satisfy a $7/8$ fraction of the clauses *in expectation* by just setting each of the n variables uniformly at random! This is because a clause with 3 literals has $2^3 - 1 = 7$ ways to be satisfied, and only one way to be unsatisfied. A deterministic polynomial-time algorithm that's *guaranteed* to satisfy at least $7/8$ of the clauses requires only a little more work.

be in P and the region known to be NP-complete. One of the major aims of contemporary research is to close those gaps, for example by proving the so-called *Unique Games Conjecture* [140].

We see a similar "invisible fence" phenomenon in Leslie Valiant's program of "accidental algorithms" [248]. The latter are polynomial-time algorithms, often for planar graph problems, that exist for certain parameter values but not for others, for reasons that are utterly opaque if one doesn't understand the strange cancellations that the algorithms exploit. A prototypical result is the following:

Theorem 19 (Valiant [248]) *Let* Planar3Sat *be the special case of* 3Sat *in which the bipartite graph of clauses and variables (with an edge between a variable and a clause whenever one occurs in the other) is planar. Now consider the following problem: given an instance of* Planar3Sat *which is monotone (i.e., has no negations), and in which each variable occurs in exactly two clauses, count the number of satisfying assignments mod k. This problem is in* P *for $k = 7$, but is* NP*-hard under randomized reductions for $k = 2$.*[18]

Needless to say (because otherwise you would've heard!), in not one of these examples have the "P region" and the "NP-complete region" of parameter space been discovered to overlap. For example, in Theorem 19, the NP-hardness proof just happens to break down if we ask about the number of solutions mod 7, the case where an algorithm is known. If $\mathsf{P} = \mathsf{NP}$ then this is, at the least, an unexplained coincidence. If $\mathsf{P} \neq \mathsf{NP}$, on the other hand, then it makes perfect sense.

3.1 Independent of Set Theory?

Since the 1970s, there's been speculation that $\mathsf{P} \neq \mathsf{NP}$ might be independent (that is, neither provable nor disprovable) from the standard axiom systems for mathematics, such as Zermelo-Fraenkel set theory. To be clear, this would mean that either

(1) a polynomial-time algorithm for NP-complete problems doesn't exist, but we can never prove it (at least not in our usual formal systems), or else

(2) a polynomial-time algorithm for NP-complete problems *does* exist, but either we can never prove that it works, or we can never prove that it halts in polynomial time.

Since $\mathsf{P} \neq \mathsf{NP}$ is an arithmetical statement (a Π_2-sentence), we can't simply excise it from mathematics, as some would do with questions in transfinite set theory, like the Continuum Hypothesis (CH) or the Axiom of Choice (AC). At the end of the day, a polynomial-time algorithm for 3Sat either exists or it doesn't! But that doesn't imply that we can prove which.

In 2003, I wrote a survey article [1] about whether $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ is formally independent, which somehow never got around to offering any opinion about the *likelihood* of that eventuality! So for the record: I regard the independence of $\mathsf{P} = \mathsf{NP}$ as a farfetched possibility, just as I do for the Riemann hypothesis, Goldbach's conjecture, and other unsolved problems of "ordinary" mathematics. At the least, I'd say that the independence of $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ has the status right now of a "free-floating speculation" with little or no support from past mathematical experience.

[18] Indeed, a natural conjecture would be that the problem is NP-hard for *all* $k \neq 7$, but this remains open (Valiant, personal communication). Note also that a problem A being "NP-hard under randomized reductions" means that all of NP is decidable by a polynomial-time randomized algorithm with an oracle for A.

There have been celebrated independence results over the past century, but as far as I know they all fall into four classes, none of which would encompass the independence of $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ from ZF set theory:

(1) Independence of statements that are themselves about formal systems: for example, that assert their own unprovability in a given system, or the system's consistency. This is the class produced by Gödel's incompleteness theorems.

(2) Independence of statements in transfinite set theory, such as CH and AC. Unlike "ordinary" mathematical statements—$\mathsf{P} \neq \mathsf{NP}$, the Riemann hypothesis, etc.—the set-theoretic ones can't be rephrased in the language of elementary arithmetic; only questions about their *provability* from various axiom systems are arithmetical. For that reason, one can question whether CH, AC, and so on need to have definite truth-values at all, independent of the axiom system. In any case, the independence of set-theoretic principles seems different in kind, and less "threatening," than the independence of arithmetical statements.[19]

(3) Independence from "weak" systems, which don't encompass all accepted mathematical reasoning. Goodstein's Theorem [101], and the non-losability of the Kirby-Paris hydra game [142], are two examples of interesting arithmetical statements that can be proved using small amounts of set theory (or ordinal induction), but not within Peano arithmetic.

(4) Independence from ZF set theory of strange combinatorial statements, which (alas) would never have been studied if not for their independence. Harvey Friedman [93] has produced striking examples of such statements.

Of course, it's possible that $\mathsf{P} \neq \mathsf{NP}$ is unprovable, but that that fact *itself* is unprovable from the axioms of set theory, and so on ad infinitum! But we can at least say that, if $\mathsf{P} \neq \mathsf{NP}$ (or for that matter, the Riemann hypothesis, Goldbach's conjecture, etc.) were *proven* independent of ZF, it would be an unprecedented development: probably history's first example of an independence result that didn't fall into one of the four classes above.[20]

The proof of independence would also have to be unlike any known independence proof. Ben-David and Halevi [41] noticed that the techniques used to prove statements such as Goodstein's Theorem independent of Peano arithmetic, actually prove independence from the stronger theory $\mathrm{PA} + \Pi_1$: that is, Peano arithmetic plus the set of all true arithmetical Π_1-sentences (sentences with a single universal quantifier and no existential quantifiers). However, if $\mathsf{P} \neq \mathsf{NP}$ could be proven independent of $\mathrm{PA} + \Pi_1$, that would mean that no Π_1-sentence of PA implying $\mathsf{P} \neq \mathsf{NP}$ could hold. And thus, for example, NP-complete problems would have to be solvable in $n^{\log \log \log \log n}$ time, and even in $n^{\alpha(n)}$ time, where $\alpha(n)$ is the inverse Ackermann function.[21] In that sense, we would "almost" have $\mathsf{P} = \mathsf{NP}$.

[19] Note also that, by the *Shoenfield absoluteness theorem* [222], one's beliefs about the Axiom of Choice, the Continuum Hypothesis, or other statements proven independent of ZF via forcing can have no effect on the provability of arithmetical statements such as $\mathsf{P} \neq \mathsf{NP}$.

[20] If a Π_1-sentence like the Goldbach Conjecture or the Riemann Hypothesis were known to be independent of ZF, then it would also be known to be *true*, since any counterexample would have a trivial finite proof! On the other hand, we could also imagine, say, the Goldbach Conjecture being proven equivalent to the consistency of ZF, in which case we could say only that *either* ZF is consistent and Goldbach is true but ZF doesn't prove either, or *else* ZF proves anything. In any case, none of this directly applies to $\mathsf{P} \neq \mathsf{NP}$, which is a Π_2-sentence.

[21] The Ackermann function, $A(n)$, can be defined as $f(n,n)$, where $f(n,k) = f(n-1, f(n,k-1))$ with boundary conditions $f(n,0) = f(n-1,1)$ and $f(0,k) = k+1$. This is one of the fastest-growing functions encountered

As Section 6 will discuss, there are various formal *barriers*—including the relativization, algebrization, and natural proofs barriers—that explain why certain existing techniques can't be powerful enough to prove $\mathsf{P} \neq \mathsf{NP}$. These barriers can be interpreted as proofs that $\mathsf{P} \neq \mathsf{NP}$ is unprovable from certain systems of axioms: namely, axioms that capture the power of the techniques in question (relativizing, algebrizing, or naturalizing techniques) [29, 123, 209].[22] In all these cases, however, the axiom systems are known not to capture all techniques in complexity theory: there are existing results that go beyond them. Thus, these barriers indicate gaps in our current techniques, rather than in the foundations of mathematics.

4 Why Is Proving $\mathsf{P} \neq \mathsf{NP}$ Difficult?

Let's suppose that $\mathsf{P} \neq \mathsf{NP}$. Then given the disarming simplicity of the statement, why is proving it so hard? As mentioned above, complexity theorists have identified three technical barriers, called *relativization* [36], *natural proofs* [210], and *algebrization* [10], that any proof of $\mathsf{P} \neq \mathsf{NP}$ will need to overcome. They've also shown that it's possible to surmount each of these barriers, though there are few results that surmount all of them simultaneously. The barriers will be discussed alongside progress toward proving $\mathsf{P} \neq \mathsf{NP}$ in Section 6.

However, we can also say something more conceptual, and possibly more illuminating, about the meta-question of why it's so hard to prove hardness. In my view, the central reason why proving $\mathsf{P} \neq \mathsf{NP}$ is hard is simply that, in case after case, there *are* amazingly clever ways to avoid brute-force search, and the diversity of those ways rivals the diversity of mathematics itself. And even if, as I said in Section 3, there seems to be an "invisible fence" separating the NP-complete problems from the slight variants of those problems that are in P—still, almost any *argument* we can imagine for why the NP-complete problems are hard would, if it worked, also apply to the variants in P.

To illustrate, we saw in Section 2.1 that 3SAT is NP-complete. We also saw that 2SAT, which is like 3SAT except with two variables per clause rather than three, is in P: indeed, 2SAT is solvable in linear time. Other variants of satisfiability that are in P include HORNSAT (where each clause is an OR of arbitrarily many non-negated variables and at most one negated variable), and XORSAT (where each clause is a linear equation mod 2, such as $x_2 \oplus x_7 \oplus x_9 \equiv 1 \pmod 2$).

Likewise, even though it's NP-complete to decide whether a given graph is 3-colorable, we can

in mathematics. Meanwhile, the *inverse* Ackermann function, $\alpha(n)$, is a monotone nondecreasing function with $\alpha(A(n)) = n$: hence, one of the slowest-growing functions one encounters.

More generally, Ben-David and Halevi [41] showed that if $\mathsf{P} \neq \mathsf{NP}$ is unprovable in $\mathrm{PA} + \Pi_1$, then NP-complete problems are solvable in $n^{f^{-1}(n)}$ time, where $f(n)$ is any function in the "Wainer hierarchy": a hierarchy of fast-growing functions, containing the Ackermann function, that can be proved to be total in PA.

By contrast, in 1969 McCreight and Meyer [168] proved a theorem one of whose striking corollaries is the following: *there exists a single computable time bound g (for which they give the algorithm), such that $\mathsf{P} = \mathsf{NP}$ if and only if 3SAT is solvable in $g(n)$ time.* Their bound $g(n)$ is "just barely" superpolynomial, and is constructed via a diagonalization procedure.

The McCreight-Meyer Theorem explains, in particular, why Ben-David and Halevi's result doesn't show that if $\mathsf{P} \neq \mathsf{NP}$ is unprovable in $\mathrm{PA} + \Pi_1$, then NP-complete problems are solvable in $n^{\beta(n)}$ time for *every* unbounded computable function β. Namely, if it showed that, then it would actually show that $\mathrm{PA} + \Pi_1$ decides the $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ problem.

[22] This is literally true for the relativization and algebrization barriers. For natural proofs, one can use the barrier to argue that $\mathsf{P} \neq \mathsf{NP}$ is unprovable from certain axioms of "bounded arithmetic," but as far as I know there's no known axiom set that *precisely* captures the power of natural proofs.

decide in linear time whether a graph is 2-colorable. Also, even though SUBSETSUM is NP-complete, we can easily decide whether there's a subset of $a_1, \ldots, a_k$ summing to b in time that's nearly linear in $a_1 + \cdots + a_k$. In other words, if each a_i is required to be encoded in "unary" notation (that is, as a list of a_i ones) rather than in binary, then SUBSETSUM is in P.

As a more interesting example, finding the maximum clique in a graph is NP-complete, as are finding the minimum vertex cover, the chromatic number, and so on. Yet in the 1960s, Edmonds [81] famously showed the following.

Theorem 20 (Edmonds [81]) *Given an undirected graph G, there's a polynomial-time algorithm to find a maximum matching: that is, a largest possible set of edges no two of which share a vertex.*

To a casual observer, matching doesn't look terribly different from the other graph optimization problems, but it *is* different.

Or consider linear, semidefinite, and convex programming. These techniques yield hundreds of optimization problems that seem similar to known NP-complete problems, and yet are solvable in P. A few examples are finding maximum flows, finding equilibria of two-player zero-sum games, training linear classifiers, and optimizing over quantum states and unitary transformations.[23]

We can also give examples of "shocking" algorithms for problems that are clearly in P. Most famously, the problem of multiplying two $n \times n$ matrices, $C = AB$, seems like it should "obviously" require $\sim n^3$ steps: $\sim n$ steps for each of the n^2 entries of the product matrix C. But in 1968, Strassen [235] discovered an algorithm that takes only $O\left(n^{\log_2 7}\right)$ steps. There's since been a long sequence of further improvements, culminating in an $O\left(n^{2.376}\right)$ algorithm by Coppersmith and Winograd [76], and its recent improvements to $O\left(n^{2.374}\right)$ by Stothers [234] and to $O\left(n^{2.373}\right)$ by Vassilevska Williams [251], with a minuscule further improvement by Le Gall [95]. Thus, letting ω be the *matrix multiplication exponent* (i.e., the least ω such that $n \times n$ matrices can be multiplied in $n^{\omega+o(1)}$ time), we know today that $\omega \in [2, 2.373]$. Some computer scientists conjecture that $\omega = 2$; but in any case, just like with attempts to prove $\mathsf{P} \neq \mathsf{NP}$, an obvious obstruction to proving $\omega > 2$ is that the proof had better *not* yield $\omega = 3$, or even a "natural-looking" bound like $\omega \geq 2.5$.

The scope of polynomial-time algorithms might seem like a trite observation, incommensurate with the challenge of explaining why it's so hard to prove $\mathsf{P} \neq \mathsf{NP}$. Yet we have evidence to the

[23] I won't have much to say about linear programming (LP) or semidefinite programming (SDP) in this survey, so perhaps this is as good a place as any to mention that today, we know a great deal about the impossibility of solving NP-complete problems in polynomial time by formulating them as "natural" LPs. This story starts in 1987, with a preprint by Swart [238] that claimed to prove $\mathsf{P} = \mathsf{NP}$ by reducing the Traveling Salesperson Problem to an LP with $O\left(n^8\right)$ variables and constraints. Swart's preprint inspired a landmark paper by Yannakakis [268] (making it possibly the most productive failed $\mathsf{P} = \mathsf{NP}$ proof in history!), in which Yannakakis showed that there is no "symmetric" LP with $n^{o(n)}$ variables and constraints that has the "Traveling Salesperson Polytope" as its projection onto a subset of the variables. This ruled out Swart's approach. Yannakakis also showed that the polytope corresponding to the maximum matching problem has no symmetric LP of subexponential size, but the polytope for the minimum spanning tree problem *does* have a polynomial-size LP. In general, expressibility by such an LP is sufficient for a problem to be in P, but not necessary.

Later, in 2012, Fiorini et al. [83] substantially improved Yannakakis's result, getting rid of the symmetry requirement. There have since been other major results in this direction: in 2014, Rothvoß [213] showed that the perfect matching polytope requires exponentially-large LPs (again with no symmetry requirement), while in 2015, Lee, Raghavendra, and Steurer [156] extended many of these lower bounds from linear to semidefinite programs.

Collectively, these results rule out one "natural" approach to proving $\mathsf{P} = \mathsf{NP}$: namely, to start from famous NP-hard optimization problems like TSP, and then find a polynomial-size LP or SDP that projects onto the polytope whose extreme points are the valid solutions. Of course, we can't yet rule out the possibility that LPs or SDPs could help prove $\mathsf{P} = \mathsf{NP}$ in some more indirect way (or via some NP-hard problem other than the specific ones that were studied); ruling *that* out seems essentially tantamount to proving $\mathsf{P} \neq \mathsf{NP}$ itself.

contrary. Over the decades, there have been hundreds of flawed proofs announced for $\mathsf{P} \neq \mathsf{NP}$. The attempt that received the most attention thus far, including coverage in *The New York Times* and other major media outlets, was that of Deolalikar [80] in 2010. But in every such case that I'm aware of, *the proof could ultimately be rejected on the ground that, if it worked, then it would also yield superpolynomial lower bounds for problems known to be in* P.

With some flawed $\mathsf{P} \neq \mathsf{NP}$ proofs, this is easy to see: for example, perhaps the author proves that 3SAT must take exponential time, by some argument that's fearsome in technical details, but ultimately boils down to "there are 2^n possible assignments to the variables, and clearly any algorithm must spend at least one step rejecting each of them." A general-purpose refutation of such arguments is simply that, if they worked, then they'd work equally well for 2SAT. Alternatively, one could point out that, as we'll see in Section 5.1, it's known how to solve 3SAT in $(4/3)^n$ time. So a $\mathsf{P} \neq \mathsf{NP}$ proof had *better* not imply a $\Omega(2^n)$ lower bound for 3SAT.

In the case of Deolalikar's $\mathsf{P} \neq \mathsf{NP}$ attempt [80], the details were more complicated, but the bottom line ended up being similar. Deolalikar appealed to certain statistical properties of the set of satisfying assignments of a *random* 3SAT instance. The claim was that, for reasons having to do with logical definability, those statistical properties precluded 3SAT from having a polynomial-time algorithm. During an intense online discussion, however, skeptics pointed out that random XORSAT—which we previously mentioned as a satisfiability variant in P—gives rise to solution sets indistinguishable from those of random 3SAT, with respect to the properties Deolalikar was using: see for example [250]. This implied that there must be one or more bugs in the proof, though it still left the task of finding them (which was done later).

None of this means that proving $\mathsf{P} \neq \mathsf{NP}$ is impossible. *A priori*, it might also have been hard to imagine a proof of the unsolvability of the halting problem, but of course we know that such a proof exists. As we'll see in Section 6.1, a central difference between the two cases is that methods from logic—namely, diagonalization and self-reference—worked to prove the unsolvability of the halting problem, but there's a precise sense in which these methods *can't* work (at least not by themselves) to prove $\mathsf{P} \neq \mathsf{NP}$. A related difference comes from the *quantitative* character of $\mathsf{P} \neq \mathsf{NP}$: somehow, any proof will need to explain why polynomial-time algorithm for 3SAT is impossible, even though a $(4/3)^n$ algorithm actually exists. In some sense, this need to make quantitative distinctions—to say that, yes, brute-force search *can* be beaten, but only by this much for this problem and by that much for that one—puts a lower bound on the sophistication of any $\mathsf{P} \neq \mathsf{NP}$ proof.

5 Strengthenings of the $\mathsf{P} \neq \mathsf{NP}$ Conjecture

I'll now survey various strengthenings of the $\mathsf{P} \neq \mathsf{NP}$ conjecture, which are often needed for applications to cryptography, quantum computing, fine-grained complexity, and elsewhere. Some of these strengthenings will play a role when, in Section 6, we discuss the main approaches to proving $\mathsf{P} \neq \mathsf{NP}$ that have been tried.

5.1 Different Running Times

There's been a great deal of progress on beating brute-force search for many NP-complete problems, even if the resulting algorithms still take exponential time. For example, Schöning proved the following in 1999.

Theorem 21 (Schöning [218]) *There's a randomized algorithm that solves* 3SAT *in* $O((4/3)^n)$ *time.*

For many NP-complete problems like HAMILTONCYCLE, for which the obvious brute-force algorithm takes $\sim n!$ time, it's also possible to reduce the running time to $O(2^n)$, or sometimes even to $O(c^n)$ for $c < 2$, through clever tricks such as *dynamic programming* (discussed in Cormen et al. [77], or any other algorithms textbook).

How far can these algorithms be pushed? For example, is it possible that 3SAT could be solved in $2^{O(\sqrt{n})}$ time, as various NP-intermediate problems like FACTORING are known to be? An important conjecture called the Exponential Time Hypothesis, or ETH, asserts that the answer is no:

Conjecture 22 (Exponential Time Hypothesis) *Any deterministic algorithm for* 3SAT *takes* $\Omega(c^n)$ *steps, for some constant* $c > 1$.

ETH is an ambitious strengthening of $\mathsf{P} \neq \mathsf{NP}$. Even assuming $\mathsf{P} \neq \mathsf{NP}$, there's by no means a consensus in the field that ETH is true—let alone still further strengthenings, like the *Strong* Exponential Time Hypothesis or SETH, which asserts that any algorithm for kSAT requires $\Omega((2-\varepsilon)^n)$ time, for some ε that goes to zero as $k \rightarrow \infty$. SETH, of course, goes even further out on a limb than ETH does, and some algorithms researchers have been actively working to disprove SETH (see [266] for example).

One thing we *do* know, however, is that if ETH or SETH hold, there are numerous implications that are not known to follow from $\mathsf{P} \neq \mathsf{NP}$ alone. One example concerns the problem of approximating the value of a "two-prover free game": here Impagliazzo, Moshkovitz, and I [9] gave an $n^{O(\log n)}$-time algorithm, but we also proved that any $f(n)$-time algorithm would imply a nearly $f\left(2^{\sqrt{n}}\right)$-time algorithm for 3SAT. Thus, assuming ETH, our quasipolynomial-time algorithm is essentially optimal, and our reduction from 3SAT to free games is *also* essentially optimal.

A second example comes from recent work of Backurs and Indyk [35], who studied the problem of EDITDISTANCE: that is, given two strings, computing the minimum number of insertions, deletions, and replacements needed to transform one string to the other. Here an $O\left(n^2\right)$ algorithm has long been known [253]. In a 2015 breakthrough, Backurs and Indyk [35] showed that algorithm to be essentially optimal, assuming SETH.

Even more recently, Abboud et al. [11] have shown that edit distance requires nearly quadratic time under a "safer" conjecture:

Theorem 23 (Abboud et al. [11]) *Suppose that the circuit satisfiability problem, for circuits of depth* $o(n)$*, can't be solved in* $(2-\varepsilon)^n$ *time for any* $\varepsilon > 0$. *Then* EDITDISTANCE *requires* $n^{2-o(1)}$ *time.*

In any case, we currently have no idea how to make similarly "fine-grained" statements about running times assuming only $\mathsf{P} \neq \mathsf{NP}$.

5.2 Nonuniform Algorithms and Circuits

$\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ asks whether there's a *single* algorithm that, for every input size n, solves an NP-complete problem like 3SAT in time polynomial in n. But we could also allow a different algorithm for each

input size. For example, it often happens in practice that a naïve algorithm works the fastest for inputs up to a certain size (say $n = 100$), then a slightly clever algorithm starts doing better, then at $n \geq 1000$ a *very* clever algorithm starts to outperform the slightly clever algorithm, and so on. In such a case, we might not even know whether the sequence terminates with a "maximally clever algorithm," or whether it goes on forever.[24]

To capture these situations, let $\mathsf{P/poly}$ be the class of languages L for which there exists a polynomial-time Turing machine M, as well as an infinite set of "advice strings" $a_1, a_2, \ldots$, where a_n is $p(n)$ bits long for some polynomial p, such that for all n and all $x \in \{0,1\}^n$, we have

$$M(x, a_n) \text{ accepts} \iff x \in L.$$

An equivalent way to define $\mathsf{P/poly}$ is as the class of languages recognized by a family of *polynomial-size circuits*, one for each input size n. In theoretical computer science, a circuit just means a directed acyclic[25] graph C_n of Boolean logic gates (such as AND, OR, NOT), with the input bits $x_1, \ldots, x_n$ at the bottom, and an output bit determining whether $x \in L$ at the top. The *size* of a circuit is the number of gates in it. The *fanin* of a circuit is the maximum number of input wires that can enter a gate g, while the *fanout* is the maximum number of output wires that can emerge from g (that is, the number of other gates that can depend directly on g's output). For now, we're considering circuits with a fanin of 2 and unlimited fanout.

We call $\mathsf{P/poly}$ the *nonuniform* generalization of P, where 'nonuniform' just means that the circuit C_n could have a different structure for each n (i.e., there need not be an efficient algorithm that outputs a description of C_n given n as input). Certainly $\mathsf{P} \subset \mathsf{P/poly}$, but there's no containment in the other direction.[26]

Now, the nonuniform version of the $\mathsf{P} \neq \mathsf{NP}$ conjecture is the following.

Conjecture 24 $\mathsf{NP} \not\subset \mathsf{P/poly}$.

If $\mathsf{P} = \mathsf{NP}$, then certainly $\mathsf{NP} \subset \mathsf{P/poly}$, but the converse need not hold. About the closest we have to a converse is the *Karp-Lipton Theorem* [135]:

Theorem 25 *If* $\mathsf{NP} \subset \mathsf{P/poly}$, *then* PH *collapses to* Σ_2^{P}.

Proof. Consider a problem in Π_2^{P}: say, "for all $x \in \{0,1\}^{p(n)}$, does there exist a $y \in \{0,1\}^{p(n)}$ such that $A(x,y)$ accepts?", for some polynomial p and polynomial-time algorithm A. Assuming $\mathsf{NP} \subset \mathsf{P/poly}$, we can solve that problem in Σ_2^{P} as follows:

[24] The so-called *Blum speedup theorem* [50] shows that we can artificially construct problems for which the sequence continues forever, there being no single fastest algorithm. No "natural" problem is known to have this behavior, though it's possible that some do. There are some natural problems, such as $\mathsf{NP} \cap \mathsf{coNP}$ and #P-complete problems, that are known *not* to have this behavior. The reason is that we can give an explicit algorithm for these problems that *must* be nearly asymptotically optimal, and that succeeds for all but finitely many input lengths n: namely, an algorithm that simulates the lexicographically-first $\log n$ Turing machines until one of them supplies a proof or interactive proof for the correct answer.

[25] Despite the term "circuit," which comes from electrical engineering, circuits in theoretical computer science are ironically *free* of cycles; they proceed from the inputs to the output via layers of logic gates.

[26] This is a rare instance where non-containment can actually be *proved*. For example, any unary language (i.e., language of the form $\{0^n : n \in S\}$) is clearly in $\mathsf{P/poly}$, since the n^{th} circuit can just hardwire whether $n \in S$. But there's an uncountable infinity of unary languages, whereas P is countable, so almost all unary languages are outside P. Alternatively, we can observe that the unary language $\left\{0^n : \text{the } n^{th} \text{ Turing machine halts}\right\}$ can't be in P, since it's simply a version of the halting problem.

- "Does there exist a circuit C such that for all x, the algorithm $A(x, C(x))$ accepts?"

For if $\mathsf{NP} \subset \mathsf{P/poly}$ and $\forall x \exists y A(x, y)$ is true, then clearly there exists a polynomial-size circuit C that takes x as input, and outputs a y such that $A(x, y)$ accepts. So we can simply use the existential quantifier in our Σ_2^{P} algorithm to guess a description of that circuit.

We conclude that, if $\mathsf{NP} \subset \mathsf{P/poly}$, then $\Pi_2^{\mathsf{P}} \subseteq \Sigma_2^{\mathsf{P}}$ (and by symmetry, $\Sigma_2^{\mathsf{P}} \subseteq \Pi_2^{\mathsf{P}}$). But this is known to cause a collapse of the entire polynomial hierarchy to Σ_2^{P}. ■

In summary, while most complexity theorists conjecture that $\mathsf{NP} \not\subset \mathsf{P/poly}$, as far as we know it's a stronger conjecture than $\mathsf{P} \neq \mathsf{NP}$. Indeed, it's even plausible that future techniques could prove $\mathsf{P} \neq \mathsf{NP}$ without proving $\mathsf{NP} \not\subset \mathsf{P/poly}$: for example, as we'll discuss in Section 6.1, we can currently prove $\mathsf{P} \neq \mathsf{EXP}$, but can't currently prove $\mathsf{EXP} \not\subset \mathsf{P/poly}$, or even $\mathsf{NEXP} \not\subset \mathsf{P/poly}$. Despite this, as we'll see in Section 6, *most* techniques that have been explored for proving $\mathsf{P} \neq \mathsf{NP}$, would actually yield the stronger result $\mathsf{NP} \not\subset \mathsf{P/poly}$ if they worked. For that reason, $\mathsf{P/poly}$ plays a central role in work on the $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ question.

There's one other aspect of circuit complexity that will play a role later in this survey: *depth*. The depth of a circuit simply means the length of the longest path from an input bit to the output bit—or, if we think of the logic gates as organized into layers, then the number of layers. There's a subclass of $\mathsf{P/poly}$ called NC^1 (the NC stands for "Nick's Class," after Nick Pippenger), which consists of all languages that are decided by a family of circuits that have polynomial size and *also* depth $O(\log n)$.[27] One can also think of NC^1 as the class of problems solvable in logarithmic time (nonuniformly) using a polynomial number of parallel processors. It's conjectured that $\mathsf{P} \not\subset \mathsf{NC}^1$ (that is, not all efficient algorithms can be parallelized), but alas, even showing $\mathsf{NEXP} \not\subset \mathsf{NC}^1$ remains open at present.

Another way to define NC^1 is as the class of languages decidable by a family of polynomial-size Boolean *formulas*. In theoretical computer science, a formula just means a circuit where every gate has fanout 1 (that is, where a gate cannot have its output fed as input to multiple other gates). To see the equivalence: in one direction, by replicating subcircuits wherever necessary, clearly any circuit of depth d and size s can be "unraveled" into a formula of depth d and size at most $2^d s$, which is still polynomial in n if $d = O(\log n)$ and $s = n^{O(1)}$. In the other direction, there's an extremely useful fact proved by Brent [56], called "depth reduction."

Proposition 26 (Brent [56]) *Given any Boolean formula of size S, there is an equivalent formula of size $S^{O(1)}$ and depth $O(\log S)$.*[28]

Because of Proposition 26, the minimum depth D of any formula for a Boolean function f is simply $\Theta(\log S)$, where S is the minimum size of any formula for f. For circuits, by contrast, size and depth are two independent variables, which might in general be related only by $D \leq S \leq 2^D$.

5.3 Average-Case Complexity

If $\mathsf{P} \neq \mathsf{NP}$, that means that there are NP problems for which no Turing machine succeeds at solving *all* instances in polynomial time. But often, especially in cryptography, we need more than that.

[27] If each logic gate depends on at most 2 inputs, then $\log_2 n$ is the smallest depth that allows the output to depend on all n input bits.

[28] Bshouty, Cleve, and Eberly [57] showed that the size of the depth-reduced formula can even be taken to be $O\left(S^{1+\varepsilon}\right)$, for any constant $\varepsilon > 0$.

It would be laughable to advertise a cryptosystem on the grounds that there *exist* messages that are hard to decode! So it's natural to ask whether there are NP problems that are hard "in the average case" or "on random instances," rather than merely in the worst case. More pointedly, does the existence of such problems follow from $\mathsf{P} \neq \mathsf{NP}$, or is it a different, stronger assumption?

The first step is to clarify what we mean by a "random instance." For some NP-complete problems, it makes sense to ask about a *uniform* random instance: for example, we can consider 3SAT with n variables and $m = \alpha n$ uniformly-random clauses (for some constant α), or 3COLORING on an Erdös-Rényi random graph.[29] In those cases, the difficulty tends to vary wildly with the problem and the precise distribution. With 3SAT, for example, if the clause/variable ratio α is too small, then random instances are trivially satisfiable, while if α is too large, then they're trivially unsatisfiable. But there's a "sweet spot," $\alpha \approx 4.25$, where random 3SAT undergoes a phase transition from satisfiable to unsatisfiable, and where the difficulty seems to blow up accordingly. Even at the threshold, however, random 3SAT might still be much easier than worst-case 3SAT: the breakthrough *survey propagation algorithm* [54] can solve random 3SAT quickly, even for α extremely close to the threshold.[30] More generally, there's been a great deal of work on understanding particular distributions over instances, often using tools from statistical physics: for an accessible introduction, see for example Moore and Mertens [172]. Unfortunately, there are almost no known reductions among these sorts of distributional problems, which would let us say that if one of them is hard then so is another. The reason is that almost any imaginable reduction from problem A to problem B will map a random instance of A to an extremely special, *non*-random instance of B.

This means that, if we want to pick random instances of NP-complete problems and be confident they're hard, then we might need carefully-tailored distributions. Levin [158], and Li and Vitányi [159], observed that there exists a "universal distribution" $\mathcal{D}$—independent of the specific problem—with the remarkable property that *any algorithm that fails on any instance, will also fail with high probability with respect to instances drawn from* $\mathcal{D}$. Briefly, one constructs $\mathcal{D}$ by giving each string $x \in \{0,1\}^*$ a probability proportional to $2^{-K(x)}$, where $K(x)$ is the *Kolmogorov complexity* of x: that is, the number of bits in the shortest computer program whose output is x. One then argues that, given any algorithm A, one can design a short computer program that brute-force searches for the first instances on which A fails—and for that reason, if there are any such instances, then $\mathcal{D}$ will assign them a high probability!

In this construction, the catch is that there's no feasible way actually to *sample* instances from the magical distribution $\mathcal{D}$. Thus, given a family of distributions $\mathcal{D} = \{\mathcal{D}_n\}_{n \geq 1}$, where $\mathcal{D}_n$ is over $\{0,1\}^{p(n)}$ (for some polynomial p), call $\mathcal{D}$ *efficiently samplable* if there exists a Turing machine that takes as input a positive integer n and a uniformly random string $r \in \{0,1\}^{q(n)}$ (for some polynomial q), and that outputs a sample from $\mathcal{D}_n$ in time polynomial in n. Then the real question, we might say, is whether there exist NP-complete problems that are hard on average with respect to efficiently samplable distributions. More formally, does the following conjecture hold?

Conjecture 27 (NP Hard on Average) *There exists a language $L \in \mathsf{NP}$, as well as an efficiently samplable family of distributions $\mathcal{D} = \{\mathcal{D}_n\}_{n \geq 1}$, such that for all polynomial-time algorithms A, there exists an n such that*

$$\Pr_{x \sim \mathcal{D}_n} [A(x) = L(x)] < 0.51.$$

[29] That is, a graph where every two vertices are connected by an edge with independent probability p.

[30] But making matters more complicated still, survey propagation fails badly on random 4SAT.

Here $L(x) \in \{0,1\}$ denotes the characteristic function of L.

Note that, if Conjecture 27 holds, then for *every* language $L' \in \mathsf{NP}$, there exists an efficiently samplable family of distributions $\mathcal{D}'$ such that L' is hard on average with respect to instances drawn from $\mathcal{D}'$. We can obtain this $\mathcal{D}'$ by simply starting with a sample from $\mathcal{D}$, and then applying the reduction from L to L'.

It's a longstanding open problem whether $\mathsf{P} \neq \mathsf{NP}$ implies Conjecture 27. There are NP-intermediate problems—one famous example being the discrete logarithm problem—that are known to have the remarkable property of *worst-case/average-case equivalence*. That is, any polynomial-time algorithm for these problems that works on (say) 10% of instances implies a polynomial-time algorithm for *all* instances; and conversely, if the problem is hard at all then it's hard on average. However, despite decades of work, no one has been able to show worst-case/average-case equivalence for any NP-complete problem (with respect to any efficiently samplable distribution), and there are known obstacles to such a result. For details, see for example the survey by Bogdanov and Trevisan [51].

5.3.1 Cryptography and One-Way Functions

One might hope that, even if we can't base secure cryptography solely on the assumption that $\mathsf{P} \neq \mathsf{NP}$, at least we could base it on Conjecture 27. But there's one more obstacle. In cryptography, we don't merely need NP problems for which it's easy to generate hard instances: rather, we need NP problems for which it's easy to generate hard instances, *along with secret solutions to those instances*. This motivates the definition of a *one-way function (OWF)*, perhaps the central concept of modern cryptography. Let $f = \{f_n\}_{n \geq 1}$ be a family of functions, with $f_n : \{0,1\}^n \rightarrow \{0,1\}^{p(n)}$ for some polynomial p. Then we call f a one-way function family if

(1) f_n is computable in time polynomial in n, but

(2) f_n is hard to invert: that is, for all polynomial-time algorithms A and polynomials q, we have

$$\Pr_{x \sim \{0,1\}^n} [f_n(A(f_n(x))) = f_n(x)] < \frac{1}{q(n)}$$

for all sufficiently large n.

We then make the following conjecture.

Conjecture 28 *There exists a one-way function family.*

Conjecture 28 is stronger than Conjecture 27, which in turn is stronger than $\mathsf{P} \neq \mathsf{NP}$. Indeed, it's not hard to show the following.

Proposition 29 *Conjecture 28 holds if and only if there exists a fast way to generate hard random* 3SAT *instances with "planted solutions": that is, an efficiently samplable family of distributions $\mathcal{D} = \{\mathcal{D}_n\}_n$ over (φ, x) pairs, where φ is a satisfiable* 3SAT *instance and x is a satisfying assignment to φ, such that for all polynomial-time algorithms A and all polynomials q,*

$$\Pr_{\varphi \sim \mathcal{D}_n} [A(\varphi) \text{ finds a satisfying assignment to } \varphi] < \frac{1}{q(n)}$$

for all sufficiently large n.

Proof. Given a one-way function family f, we can generate a hard random 3SAT instance with a planted solution by choosing $x \in \{0,1\}^n$ uniformly at random, computing $f_n(x)$, and then using the Cook-Levin Theorem (Theorem 2) to construct a 3SAT instance that encodes the problem of finding a preimage of $f_n(x)$. Conversely, given a polynomial-time algorithm that takes as input a positive integer n and a random string $r \in \{0,1\}^{p(n)}$ (for some polynomial p), and that outputs a hard 3SAT instance φ_r together with a planted solution x_r to φ_r, the function $f_n(r) := \varphi_r$ will necessarily be one-way, since inverting f_n would let us find a satisfying assignment to φ_r. ■

Conjecture 28 turns out to suffice for building most of the ingredients of private-key cryptography, notably including pseudorandom generators [116] and pseudorandom functions [98]. Furthermore, while Conjecture 28 is formally stronger than $\mathsf{P} \neq \mathsf{NP}$, Proposition 29 suggests that the two conjectures are conceptually similar: "all we're asking for" is a hard NP problem, together with a fast way to generate hard solved instances of it!

This contrasts with the situation for *public-key* cryptography—i.e., the kind of cryptography that doesn't require any secrets to be shared in advance, and which is used for sending credit-card numbers over the web. To create a secure public-key cryptosystem, we need something even stronger than Conjecture 28: for example, a *trapdoor* OWF,[31] which is an OWF with the additional property that it becomes easy to invert if we're given a secret "trapdoor" string generated along with the function. We do, of course, have candidates for secure public-key cryptosystems, which are based on problems such as factoring, discrete logarithms (over both multiplicative groups and elliptic curves), and finding short nonzero vectors in lattices. To date, however, all public-key cryptosystems require "sticking our necks out," and conjecturing the hardness of some specific NP-intermediate problem, something with much more structure than any known NP-complete problem.

In other words, for public-key cryptography, today one has to make conjectures that go fundamentally beyond $\mathsf{P} \neq \mathsf{NP}$, or even the existence of OWFs. Even if someone proved $\mathsf{P} \neq \mathsf{NP}$ or Conjecture 28, reasonable doubt would still remain about the security of known public-key cryptosystems.

5.4 Randomized Algorithms

Even assuming $\mathsf{P} \neq \mathsf{NP}$, we can still ask whether NP-complete problems can be solved in polynomial time with help from random bits. This is a different question than whether NP is hard on average: whereas before we were asking about algorithms that solve *most* instances (with respect to some distribution), now we're asking about algorithms that solve *all* instances, for *most* choices of some auxiliary random numbers.

Historically, algorithm designers have often resorted to randomness, to deal with situations where *most* choices that an algorithm could make are fine, but any *specific* choice will lead to terrible behavior on certain inputs. For example, in Monte Carlo simulation, used throughout science and engineering, we estimate the volume of a high-dimensional object by just sampling random points, and then checking what fraction of them lie inside. Another example concerns *primality testing*: that is, deciding the language

$$\text{PRIMES} = \{N : N \text{ is a binary encoding of a prime number}\}.$$

In modern cryptosystems such as RSA, generating a key requires choosing large random primes, so it's just as important that primality testing be *easy* as that the related factoring problem be hard!

[31] There are closely-related objects, such as "lossy" trapdoor OWFs (see [196]), that also suffice for building public-key cryptosystems.

In the 1970s, Rabin [200] and Solovay and Strassen [230] showed how to decide PRIMES in time polynomial in $\log N$ (i.e., the number of digits of N). The catch was that their algorithms were randomized: in addition to N, they required a second input r; and for each N, the algorithms were guaranteed to succeed for most r's but not all of them. Miller [171] also proposed a deterministic polynomial-time algorithm for PRIMES, but could only prove the algorithm correct assuming the Extended Riemann Hypothesis. Finally, after decades of work on the problem, in 2002 Agrawal, Kayal, and Saxena [15] gave an unconditional proof that PRIMES is in P. In other words, if we only care about testing primality in polynomial time, and not about the degree of the polynomial, then randomness was never needed after all.

A third example of the power of randomness comes from the *polynomial identity testing* (PIT) problem. Here we're given as input a circuit or formula, composed of addition and multiplication gates, that computes a polynomial $p : \mathbb{F} \rightarrow \mathbb{F}$ over a finite field $\mathbb{F}$. The question is whether p is the identically-zero polynomial—that is, whether the identity $p(x) = 0$ holds. If $\deg(p) \ll |\mathbb{F}|$, then the Fundamental Theorem of Algebra immediately suggests a way to solve this problem: simply pick an $x \in \mathbb{F}$ uniformly at random and check whether $p(x) = 0$. Since a nonzero polynomial p can vanish on at most $\deg(p)$ points, the probability that we'll "get unlucky" and choose one of those points is at most $\deg(p) / |\mathbb{F}|$. To this day, no one knows of any deterministic approach that achieves similar performance.[32] Derandomizing PIT—that is, replacing the randomized algorithm by a comparably-efficient deterministic one—is considered one of the frontier problems of theoretical computer science. For details, see for example the survey of Shpilka and Yehudayoff [225].

5.4.1 BPP and Derandomization

What's the power of randomness more generally? Can *every* randomized algorithm be derandomized, as ultimately happened with PRIMES? To explore these issues, complexity theorists study several randomized generalizations of the class P. We'll consider just one of them: *Bounded-Error Probabilistic Polynomial-Time*, or BPP, is the class of languages $L \subseteq \{0,1\}^*$ for which there exists a polynomial-time Turing machine M, as well as a polynomial p, such that for all inputs $x \in \{0,1\}^n$,

$$\Pr_{r \in \{0,1\}^{p(n)}} [M(x,r) = L(x)] \geq \frac{2}{3}.$$

In other words, for every x, the machine M must correctly decide whether $x \in L$ "most of the time" (that is, for most choices of r). Crucially, here we can easily replace the constant $2/3$ by any other number between $1/2$ and 1, or even by a function like $1 - 2^{-n}$. So for example, if we wanted to know $x \in L$ with 0.999999 confidence, then we'd simply run M several times, with different independent values of r, and then output the majority vote among the results.

It's clear that $\mathsf{P} \subseteq \mathsf{BPP} \subseteq \mathsf{PSPACE}$. More interestingly, Sipser [226] and Lautemann [155] proved that BPP is contained in $\Sigma_2^{\mathsf{P}} \cap \Pi_2^{\mathsf{P}}$ (that is, the second level of PH). The Rabin-Miller and Solovay-Strassen algorithms imply that PRIMES $\in$ BPP.

Today, most complexity theorists conjecture that what happened to PRIMES can happen to all of BPP:

Conjecture 30 $\mathsf{P} = \mathsf{BPP}$.

[32] At least, not for *arbitrary* polynomials computed by small formulas or circuits. A great deal of progress has been made derandomizing PIT for restricted classes of polynomials. In fact, the deterministic primality test of Agrawal, Kayal, and Saxena [15] was based on a derandomization of one extremely special case of PIT.

The reason for this conjecture is that it follows from the existence of good enough *pseudorandom generators*, which we could use to replace the random string r in any BPP algorithm M by deterministic strings that "look random, as far as M can tell." Furthermore, work in the 1990s showed that, if we grant certain plausible lower bounds on circuit size, then these pseudorandom generators exist. Perhaps the most striking result along these lines is that of Impagliazzo and Wigderson [126]:

Theorem 31 (Impagliazzo-Wigderson [126]) *Suppose there exists a language decidable in 2^n time, which requires nonuniform circuits of size $2^{\Omega(n)}$. Then* $\mathsf{P} = \mathsf{BPP}$.

Of course, if $\mathsf{P} = \mathsf{BPP}$, then the question of whether randomized algorithms can efficiently solve NP-complete problems is just the original $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ question in a different guise. Ironically, however, the "obvious" approach to proving $\mathsf{P} = \mathsf{BPP}$ is to prove a strong circuit lower bound—and if we knew how to do that, perhaps we could prove $\mathsf{P} \neq \mathsf{NP}$ as well!

Even if we don't assume $\mathsf{P} = \mathsf{BPP}$, it's easy to show that deterministic *nonuniform* algorithms (see Section 5.2) can simulate randomized algorithms:

Proposition 32 (Adleman [12]) $\mathsf{BPP} \subset \mathsf{P/poly}$.

Proof. Let the language L be decided by a BPP algorithm that uses $p(n)$ random bits. Then by using $q(n) = O(n \cdot p(n))$ random bits, running the algorithm $O(n)$ times with independent random bits each time, and outputting the majority answer, we can push the probability of error on any given input $x \in \{0,1\}^n$ from $1/3$ down to (say) 2^{-2n}. Thus, the probability that there *exists* an $x \in \{0,1\}^n$ on which the algorithm errs is at most $2^n \left(2^{-2n}\right) = 2^{-n}$. This means, in particular, that there must be a fixed choice for the random string $r \in \{0,1\}^{q(n)}$ that causes the algorithm to succeed on all $x \in \{0,1\}^n$. So to decide L in P/poly, we simply "hardwire" that r as the advice. ■

By combining Theorem 25 with Proposition 32, we immediately obtain that if $\mathsf{NP} \subseteq \mathsf{BPP}$, then the polynomial hierarchy collapses to the second level. So the bottom line is that the $\mathsf{NP} \subseteq \mathsf{BPP}$ question is likely identical to the $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ question, but is extremely tightly related even if not.

5.5 Quantum Algorithms

The class BPP might not exhaust what the physical world lets us efficiently compute, with quantum computing an obvious contender for going further. In 1993, Bernstein and Vazirani [45] defined the complexity class BQP, or Bounded-Error Quantum Polynomial-Time, as a quantum-mechanical generalization of BPP. (Details of quantum computing and BQP are beyond the scope of this survey, but see [190, 6].) Bernstein and Vazirani, along with Adleman, DeMarrais, and Huang [13], also showed some basic containments:

$$\mathsf{P} \subseteq \mathsf{BPP} \subseteq \mathsf{BQP} \subseteq \mathsf{PP} \subseteq \mathsf{P}^{\#\mathsf{P}} \subseteq \mathsf{PSPACE}.$$

In 1994, Shor [223] famously showed that the factoring and discrete logarithm problems are in BQP—and hence, that a scalable quantum computer, if built, could break almost all currently-used public-key cryptography. To design his quantum algorithms, Shor had to exploit extremely special properties of factoring and discrete logarithm, which aren't known to hold for NP-complete problems.

The quantum analogue of the $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ question is the question of whether $\mathsf{NP} \subseteq \mathsf{BQP}$: that is, *can quantum computers solve* NP*-complete problems in polynomial time?*[33] Most quantum computing researchers conjecture that the answer is no:

Conjecture 33 $\mathsf{NP} \not\subset \mathsf{BQP}$.

Naturally, there's little hope of proving Conjecture 33 at present, since any proof would imply $\mathsf{P} \neq \mathsf{NP}$! We don't even know today how to prove conditional statements (analogous to what we have for BPP and $\mathsf{P/poly}$): for example, that if $\mathsf{NP} \subseteq \mathsf{BQP}$ then PH collapses. On the other hand, it *is* known that, if a fast quantum algorithm for NP-complete problems exists, then in some sense it will have to be extremely different from Shor's or any other known quantum algorithm. For example, Bennett et al. [43] showed that, if we ignore the structure of NP-complete problems, and just consider the abstract task of searching an unordered list, then quantum computers can provide at most a square-root speedup over the classical running time. As Bennett et al. [43] noted, this implies that there exists an oracle A such that $\mathsf{NP}^A \not\subset \mathsf{BQP}^A$. Note that the square-root speedup is actually achievable, using *Grover's algorithm* [107]. For most NP-complete problems, however, the fastest known quantum algorithm will be obtained by simply layering Grover's algorithm on top of the fastest known classical algorithm, yielding a quadratic speedup but no more.[34] So for example, as far as anyone knows today, even a quantum computer would need $2^{\Omega(n)}$ time to solve 3Sat.

Of course, one can also wonder whether the physical world might provide computational resources even *beyond* quantum computing (based on black holes? closed timelike curves? modifications to quantum mechanics?), and if so, whether *those* resources might enable the polynomial-time solution of NP-complete problems. Such speculations are beyond the scope of this article, but see for example [3].

6 Progress

One common view among mathematicians is that questions like $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$, while undoubtedly important, are just too hard to make progress on in the present state of mathematics. It's true that we seem to be nowhere close to a solution, but in this section, I'll build a case that the extreme pessimistic view is unwarranted. I'll explain what genuine knowledge I think we have, relevant to

[33] One can also consider the QMA-complete problems, which are a quantum generalization of the NP-complete problems themselves (see [52]), but we won't pursue that here.

[34] One can artificially design an NP-complete problem with a superpolynomial quantum speedup over the best known classical algorithm by, for example, taking the language

$$L = \left\{0\varphi 0 \cdots 0 \mid \varphi \text{ is a satisfiable 3Sat instance of size } n^{0.01}\right\} \cup$$
$$\left\{1x \mid x \text{ is a binary encoding of a positive integer with an odd number of distinct prime factors}\right\}.$$

Clearly L is NP-complete, and a quantum algorithm can decide L in $O\left(c^{n^{0.01}}\right)$ time for some c, whereas the best known classical algorithm will take $\sim \exp\left(n^{1/3}\right)$ time.

Conversely, there are also NP-complete problems with no significant quantum speedup known—say, because the best known classical algorithm is based on dynamic programming, and it's unknown how to combine that with Grover's algorithm. A candidate example is the Traveling Salesperson Problem, which is solvable in $O\left(2^n \operatorname{poly}(n)\right)$ time using the Held-Karp dynamic programming algorithm [118], whereas Grover's algorithm seems to yield only the worse bound $O(\sqrt{n!})$.

proving $\mathsf{P} \neq \mathsf{NP}$, that we didn't have thirty years ago or in many cases ten years ago. One could argue that, if $\mathsf{P} \neq \mathsf{NP}$ is a distant peak, then all the progress has remained in the foothills. On the other hand, scaling the foothills has *already* been nontrivial, so anyone aiming for the summit had better get acquainted with what's been done.

More concretely, I'll tell a story of the interaction between *lower bounds* and *barriers*: on the one hand, actual successes in proving superpolynomial or exponential lower bounds in interesting models of computation; but on the other, explanations for why the techniques used to achieve those successes don't extend to prove $\mathsf{P} \neq \mathsf{NP}$. We'll see how the barriers influence the next generation of lower bound techniques, which are sometimes specifically designed to evade the barriers, or evaluated on their potential to do so.

With a single exception—namely, the Mulmuley-Sohoni Geometric Complexity Theory program—I'll restrict my narrative to ideas that have already had definite successes in proving new limits on computation. The drawback of this choice is that in many cases, the ideas that are concrete enough to have worked for *something*, are also concrete enough that we understand why they can't work for $\mathsf{P} \neq \mathsf{NP}$! My defense is that this section would be unmanageably long, if it had to cover *every* idea about how $\mathsf{P} \neq \mathsf{NP}$ might someday be proved.

I should, however, at least mention some important approaches to lower bounds that will be missing from my subsequent narrative. The first is *descriptive complexity theory*; see for example the book of Immerman [122] for a good introduction. Descriptive complexity characterizes many complexity classes in terms of their logical expressive power: for example, P corresponds to sentences expressible in first-order logic with linear order and a least fixed point; NP to sentences expressible in existential second-order logic; PSPACE to sentences expressible in second-order logic with transitive closure; and EXP to sentences expressible in second-order logic with a least fixed point. The hope is that characterizing complexity classes in this way, with no explicit mention of resource bounds, will make it easier to see which are equal and which different. There's one major piece of evidence for this hope: namely, descriptive complexity played an important role in the proof by Immerman [121] that nondeterministic space is closed under complement (though the independent proof by Szelepcsényi [239] of the same result didn't use these ideas). Descriptive complexity theory hasn't yet led to new separations between complexity classes.

The second approach is *lower bounds via communication complexity*. Given a Boolean function $f : \{0,1\}^n \rightarrow \{0,1\}$, consider the following communication game: Alice receives an n-bit input $x = x_1 \cdots x_n$ such that $f(x) = 0$, Bob receives an input y such that $f(y) = 1$, and their goal is to agree on an index $i \in \{1, \ldots, n\}$ such that $x_i \neq y_i$. Let C_f be the communication complexity of this game: that is, the minimum number of bits that Alice and Bob need to exchange to win the game, if they use an optimal protocol (and where the communication cost is maximized over all x, y pairs). Then in 1990, Karchmer and Wigderson [133] showed the following remarkable connection.

Theorem 34 (Karchmer-Wigderson [133]) *For any f, the minimum depth of any Boolean circuit for f is equal to C_f.*

Combined with Proposition 26 (depth-reduction for formulas), Theorem 34 implies that every Boolean function f requires formulas of size at least 2^{C_f}: in other words, *communication lower bounds imply formula-size lower bounds*. Now, communication complexity is a well-established area of theoretical computer science with many strong lower bounds; see for example the book by Kushilevitz and Nisan [148]. Thus, we might hope that lower-bounding the communication cost

of the "Karchmer-Wigderson game," say for an NP-complete problem like HamiltonCycle, could be a viable approach to proving $\mathsf{NP} \not\subset \mathsf{NC}^1$, which in turn would be a huge step toward $\mathsf{P} \neq \mathsf{NP}$.

See Section 6.2.2 for Karchmer and Wigderson's applications of a similar connection to *monotone* formula-size lower bounds. Also see Aaronson and Wigderson [10] for further connections between communication complexity and computational complexity, including even a "communication complexity lower bound" that if true would imply $\mathsf{P} \neq \mathsf{NP}$. Of course, the question is whether these translations merely shift the difficulty of complexity class separations to a superficially different setting, or whether they set the stage for genuinely new insights.

The third approach is *lower bounds via derandomization*. In Section 5.4.1, we discussed the discovery in the 1990s that, if sufficiently strong circuit lower bounds hold, then $\mathsf{P} = \mathsf{BPP}$: that is, every randomized algorithm can be made deterministic with only a polynomial slowdown. In the early 2000s, it was discovered that converse statements often hold as well: that is, *derandomizations of randomized algorithms imply circuit lower bounds*. Probably the best-known result along these lines is that of Kabanets and Impagliazzo [130]:

Theorem 35 ([130]) *Suppose the polynomial identity testing problem from Section 5.4 is in* P. *Then either* $\mathsf{NEXP} \not\subset \mathsf{P/poly}$, *or else the permanent function has no polynomial-size arithmetic circuits (see Section 6.5.1).*

As usual, the issue is that it's not clear whether we should interpret this result as giving a plausible path toward proving circuit lower bounds (namely, by derandomizing PIT), or simply as explaining why derandomizing PIT will be hard (namely, because doing so will imply circuit lower bounds)! In any case, Sections 6.3.2 and 6.4.2 will give further examples where derandomization results would imply new circuit lower bounds.

The fourth approach could be called *lower bounds via "innocent-looking" combinatorics problems*. Here's an example: given an $n \times n$ matrix A, say over the finite field $\mathbb{F}_2$, call A *rigid* if not only does A have rank $\Omega(n)$, but any matrix obtained by changing $O\left(n^{1/10}\right)$ entries in each row of A also has rank $\Omega(n)$. It's easy to show, via a counting argument, that almost all matrices $A \in \mathbb{F}_2^{n \times n}$ are rigid. On the other hand, Valiant [245] made the following striking observation in 1977: if we manage to find any *explicit example* of a rigid matrix, then we also get an explicit example of a Boolean function that can't be computed by any circuit of linear size and logarithmic depth.

For another connection in the same spirit, given a 3-dimensional tensor $A \in \mathbb{F}_2^{n \times n \times n}$, let the *rank* of A be the smallest r such that A can be written as the sum of r rank-one tensors (that is, tensors of the form $t_{ijk} = x_i y_j z_k$). Then it's easy to show, via a counting argument, that almost all tensors $A \in \mathbb{F}_2^{n \times n \times n}$ have rank $\Omega\left(n^2\right)$. On the other hand, Strassen [236] observed in 1973 that, if we find any explicit example of a 3-dimensional tensor with rank r, then we also get an explicit example of a Boolean function with circuit complexity $\Omega(r)$.[35] Alas, proving that any explicit matrix is rigid, or that any explicit tensor has superlinear rank, have turned out to be staggeringly hard problems—as perhaps shouldn't surprise us, given the implications for circuit lower bounds!

[35] Going even further, Raz [203] proved in 2010 that, if we manage to show that any explicit d-dimensional tensor $A : [n]^d \rightarrow \mathbb{F}$ has rank at least $n^{d(1-o(1))}$, then we've also shown that the $n \times n$ permanent function has no polynomial-size arithmetic formulas. It's easy to construct explicit d-dimensional tensors with rank $n^{\lfloor d/2 \rfloor}$, but the current record is an explicit d-dimensional tensor with rank at least $2n^{\lfloor d/2 \rfloor} + n - O(d \log n)$ [19].

Note that, if we could show that the permanent had no $n^{O(\log n)}$-size arithmetic formulas, that would imply Valiant's famous Conjecture 70: that the permanent has no polynomial-size arithmetic *circuits*. However, Raz's technique seems incapable of proving formula-size lower bounds better than $n^{\Omega(\log \log n)}$.

The rest of the section is organized as follows:

- Section 6.1 covers logical techniques, which typically fall prey to the relativization barrier.
- Section 6.2 covers combinatorial techniques, which typically fall prey to the natural proofs barrier.
- Section 6.3 covers "hybrid" techniques (logic plus arithmetization), many of which fall prey to the algebrization barrier.
- Sections 6.4 covers "ironic complexity theory" (as exemplified by the recent work of Ryan Williams), or the use of nontrivial algorithms to prove circuit lower bounds.
- Section 6.5 covers arithmetic circuit lower bounds, which *probably* fall prey to arithmetic variants of the natural proofs barrier (though this remains disputed).
- Section 6.6 covers Mulmuley and Sohoni's Geometric Complexity Theory (GCT), an audacious program to tackle $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ and related problems by reducing them to questions in algebraic geometry and representation theory (and which is also an example of "ironic complexity theory").

Note that, for the approaches covered in Sections 6.4 and 6.6, no formal barriers are yet known.

6.1 Logical Techniques

In the 1960s, Hartmanis and Stearns [112] realized that, by simply "scaling down" Turing's diagonalization proof of the undecidability of the halting problem, we can at least prove *some* separations between complexity classes. In particular, we can generally show that more of the same resource (time, memory, etc.) lets us decide more languages than less of that resource. Here's a special case of their so-called *Time Hierarchy Theorem*.

Theorem 36 (Hartmanis-Stearns [112]) P *is strictly contained in* EXP.

Proof. Let

$$L = \{(\langle M \rangle, x, 0^n) : M(x) \text{ halts in at most } 2^n \text{ steps}\}.$$

Clearly $L \in \mathsf{EXP}$. On the other hand, suppose by contradiction that $L \in \mathsf{P}$. Then there's some polynomial-time Turing machine A such that $A(z)$ accepts if and only if $z \in L$. Let A run in $p(n + |\langle M \rangle| + |x|)$ time. Then using A, we can easily produce another machine B that does the following:

- Takes input $(\langle M \rangle, 0^n)$.
- Runs forever if $M(\langle M \rangle, 0^n)$ halts in at most 2^n steps; otherwise halts.

Note that, if B halts at all, then it halts after only $p(2n + 2|\langle M \rangle|) = n^{O(1)}$ steps.

Now consider what happens when B is run on input $(\langle B \rangle, 0^n)$. If $B(\langle B \rangle, 0^n)$ runs forever, then $B(\langle B \rangle, 0^n)$ halts. Conversely, if $B(\langle B \rangle, 0^n)$ halts, then for all sufficiently large n, it halts in fewer than 2^n steps, but that means that $B(\langle B \rangle, 0^n)$ runs forever. So we conclude that B, and hence A, can't have existed. ■

More broadly, the same argument shows that there are languages decidable in $O\left(n^2\right)$ time but not in $O\left(n\right)$ time, in $O\left(n^{100}\right)$ time but not in $O\left(n^{99}\right)$ time, and so on for almost every natural pair of runtime bounds. (Technically, we have $\mathsf{TIME}\left(f\left(n\right)\right) \neq \mathsf{TIME}\left(g\left(n\right)\right)$ for every f, g that are *time-constructible*—that is, there exist Turing machines that run for $f\left(n\right)$ and $g\left(n\right)$ steps given n as input—and that are separated by more than a $\log n$ multiplicative factor.) Likewise, the *Space Hierarchy Theorem* shows that there are languages decidable in $O\left(f\left(n\right)\right)$ space but not in $O\left(g\left(n\right)\right)$ space, for all natural $f\left(n\right) \gg g\left(n\right)$. Cook [74] also proved a hierarchy theorem for the nondeterministic time classes, which will play an important role in Section 6.4:

Theorem 37 (Nondeterministic Time Hierarchy Theorem [74]) *For all time-constructible f, g such that $f\left(n+1\right) = o\left(g\left(n\right)\right)$, we have that $\mathsf{NTIME}\left(f\left(n\right)\right)$ is strictly contained in $\mathsf{NTIME}\left(g\left(n\right)\right)$.*

One amusing consequence of the hierarchy theorems is that we can prove, for example, that $\mathsf{P} \neq \mathsf{SPACE}\left(n\right)$, even though we can't prove either that $\mathsf{P} \not\subset \mathsf{SPACE}\left(n\right)$ or that $\mathsf{SPACE}\left(n\right) \not\subset \mathsf{P}$! For suppose by contradiction that $\mathsf{P} = \mathsf{SPACE}\left(n\right)$. Then by a padding argument (cf. Proposition 17), P would also contain $\mathsf{SPACE}\left(n^2\right)$, and therefore equal $\mathsf{SPACE}\left(n^2\right)$. But then we'd have $\mathsf{SPACE}\left(n\right) = \mathsf{SPACE}\left(n^2\right)$, violating the Space Hierarchy Theorem. Most computer scientists conjecture both that $\mathsf{P} \not\subset \mathsf{SPACE}\left(n\right)$ *and* that $\mathsf{SPACE}\left(n\right) \not\subset \mathsf{P}$, but proving either statement by itself is a much harder problem.

In summary, there really is a rich, infinite hierarchy of harder and harder computable problems. Complexity classes don't collapse in the most extreme ways imaginable, with (say) everything solvable in linear time.

6.1.1 Circuit Lower Bounds Based on Counting

A related idea—not exactly "diagonalization," but counting arguments made explicit—can also be used to show that certain problems can't be solved by polynomial-size *circuits*. This story starts with Claude Shannon [221], who made the following fundamental observation in 1949.

Proposition 38 (Shannon [221]) *There exists a Boolean function $f : \{0,1\}^n \rightarrow \{0,1\}$, on n variables, such that any circuit to compute f requires at least $\Omega\left(2^n/n\right)$ logic gates. Indeed, almost all Boolean functions on n variables (that is, a $1 - o\left(1\right)$ fraction of them) have this property.*[36]

Proof. There are 2^{2^n} different Boolean functions f on n variables, but only

$$\sum_{t=1}^{T} \binom{n}{2} \binom{n+1}{2} \cdots \binom{n+t-1}{2} < \left(n+T\right)^{2T}$$

different Boolean circuits with n inputs and at most T NAND gates. Since each circuit only represents one function, and since $\left(n+T\right)^{2T} = o\left(2^{2^n}\right)$ when $T = o\left(2^n/n\right)$, it follows by a counting argument (i.e., the pigeonhole principle) that *some* f must require a circuit with $T = \Omega\left(2^n/n\right)$ NAND gates—and indeed, that almost all of the 2^{2^n} possible f's must have this property. The number of AND, OR, and NOT gates required is related to the number of NAND gates by a constant factor, so is also $\Omega\left(2^n/n\right)$. ■

[36] With some effort, Shannon's lower bound can be shown to be tight: that is, every n-variable Boolean function *can* be represented by a circuit of size $O\left(2^n/n\right)$. (The obvious upper bound is $O\left(n2^n\right)$.)

Famously, Proposition 38 shows that there *exist* Boolean functions that require exponentially large circuits—in fact, that almost all of them do—yet it fails to produce a single example of such a function! It tells us nothing whatsoever about 3SAT or CLIQUE or any other particular function that might interest us. In that respect, it's similar to Shannon's celebrated proof that almost all codes are good error-correcting codes, which also fails to produce a single example of such a code. Just like, in the decades after Shannon, the central research agenda of coding theory was to "make Shannon's argument explicit" by finding *specific* good error-correcting codes, so too the agenda of circuit complexity has been to "make Proposition 38 explicit" by finding specific functions that provably require large circuits.

In some cases, the mere fact that we know, from Proposition 38, that hard functions *exist* lets us "bootstrap" to show that particular complexity classes must contain hard functions. Here's an example of this.

Theorem 39 $\mathsf{EXPSPACE} \not\subset \mathsf{P/poly}$.

Proof. Let n be sufficiently large. Then by Proposition 38, there exist functions $f : \{0,1\}^n \rightarrow \{0,1\}$ with circuit complexity at least $c2^n/n$, for some constant $c > 0$. Thus, if we list all the 2^{2^n} functions in lexicographic order by their truth tables, there must be a first function in the list, call it f_n, with circuit complexity at least $c2^n/n$. We now define

$$L := \bigcup_{n \geq 1} \{x \in \{0,1\}^n : f_n(x) = 1\}.$$

Then by construction, $L \notin \mathsf{P/poly}$. On the other hand, enumerating all n-variable Boolean functions, calculating the circuit complexity of each, and finding the first one with circuit complexity at least $c2^n/n$ can all be done in exponential space. Hence $L \in \mathsf{EXPSPACE}$. ■

There's also a "scaled-down version" of Theorem 39, proved in the same way:

Theorem 40 *For every fixed k, there is a language in* PSPACE *that does not have circuits of size n^k.*[37]

By being a bit more clever, Kannan [132] lowered the complexity class in Theorem 39 from EXPSPACE to $\mathsf{NEXP}^{\mathsf{NP}}$.

Theorem 41 (Kannan [132]) $\mathsf{NEXP}^{\mathsf{NP}} \not\subset \mathsf{P/poly}$.

Proof. First, we claim that $\mathsf{EXP}^{\mathsf{NP}^{\mathsf{NP}}} \not\subset \mathsf{P/poly}$. The reason is simply a more careful version of the proof of Theorem 39: in $\mathsf{EXP}^{\mathsf{NP}^{\mathsf{NP}}}$, we can do an explicit binary search for the lexicographically first Boolean function $f_n : \{0,1\}^n \rightarrow \{0,1\}$ such that every circuit of size at most (say) $c2^n/n$ disagrees with f_n on some input x. (Such an f_n must exist by a counting argument.)

Next, suppose by contradiction that $\mathsf{NEXP}^{\mathsf{NP}} \subset \mathsf{P/poly}$. Then certainly $\mathsf{NP} \subset \mathsf{P/poly}$. By the Karp-Lipton Theorem (Theorem 25), this implies that $\mathsf{PH} = \Sigma_2^{\mathsf{P}}$, so in particular $\mathsf{P}^{\mathsf{NP}^{\mathsf{NP}}} = \mathsf{NP}^{\mathsf{NP}}$. By upward translation (as in Proposition 17), this in turn means that $\mathsf{EXP}^{\mathsf{NP}^{\mathsf{NP}}} = \mathsf{NEXP}^{\mathsf{NP}}$. But we already know that $\mathsf{EXP}^{\mathsf{NP}^{\mathsf{NP}}}$ doesn't have polynomial-size circuits, and therefore neither does $\mathsf{NEXP}^{\mathsf{NP}}$. ■

[37] Crucially, this will be a different language for each k; otherwise we'd get $\mathsf{PSPACE} \not\subset \mathsf{P/poly}$, which is far beyond our current ability to prove.

Amusingly, if we work out the best possible lower bound that we can get from Theorem 41 on the circuit complexity of a language in $\mathsf{NEXP}^{\mathsf{NP}}$, it turns out to be *half-exponential*: that is, a function f such that $f(f(n))$ grows exponentially. Such functions exist, but have no closed-form expressions.

Directly analogous to Theorem 40, a "scaled-down" version of the proof of Theorem 41 shows that, for every fixed k, there's a language in $\Sigma_2^{\mathsf{P}} = \mathsf{NP}^{\mathsf{NP}}$ that doesn't have circuits of size n^k.

In Section 6.3, we'll discuss slight improvements to these results that can be achieved with algebraic methods. Nevertheless, it (sadly) remains open even to show that $\mathsf{NEXP} \not\subset \mathsf{P/poly}$, or that there's a language in NP that doesn't have linear-sized circuits.

6.1.2 The Relativization Barrier

The magic of diagonalization, self-reference, and counting arguments is how abstract and general they are: they never require us to "get our hands dirty" by understanding the inner workings of algorithms or circuits. But as was recognized early in the history of complexity theory, the price of generality is that the logical techniques are extremely limited in scope.

Often the best way to understand the limits of a proposed approach for proving a statement S, is to examine *what else besides* S the approach would prove if it worked—i.e., which stronger statements S' the approach "fails to differentiate" from S. If any of the stronger statements are false, then the approach can't prove S either.

That is exactly what Baker, Gill, and Solovay [36] did for diagonalization in 1975, when they articulated the *relativization barrier*. Their central insight was that almost all the techniques we have for proving statements in complexity theory—such as $\mathcal{C} \subseteq \mathcal{D}$ or $\mathcal{C} \not\subset \mathcal{D}$, where $\mathcal{C}$ and $\mathcal{D}$ are two complexity classes—are so general that, if they work at all, then they actually prove $\mathcal{C}^A \subseteq \mathcal{D}^A$ or $\mathcal{C}^A \not\subset \mathcal{D}^A$ *for all possible oracles* A. In other words: if all the machines that appear in the proof are enhanced in the same way, by being given access to the same oracle, the proof is completely oblivious to that change, and goes through just as before. A proof with this property is said to "relativize," or to hold "in all possible relativized worlds" or "relative to any oracle."

Why do so many proofs relativize? Intuitively, because the proofs only do things like using one Turing machine M_1 to simulate a second Turing machine M_2 step-by-step, without examining either machine's internal structure. In that case, if M_2 is given access to an oracle A, then M_1 can still simulate M_2 just fine, provided that M_1 is *also* given access to A, in order to simulate M_2's oracle calls.

To illustrate, you might want to check that the proofs of Theorems 36, 39, and 41 can be straightforwardly modified to show that, more generally:

- $\mathsf{P}^A \neq \mathsf{EXP}^A$ for all oracles A.
- $\mathsf{EXPSPACE}^A \not\subset \mathsf{P}^A/\mathsf{poly}$ for all oracles A.
- $\mathsf{NEXP}^{\mathsf{NP}^A} \not\subset \mathsf{P}^A/\mathsf{poly}$ for all oracles A.

Alas, Baker, Gill, and Solovay then observed that no relativizing technique can possibly resolve the $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ question. For, unlike (say) $\mathsf{P} \stackrel{?}{=} \mathsf{EXP}$ or the unsolvability of the halting problem, $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ admits "contradictory relativizations": there are some oracle worlds where $\mathsf{P} = \mathsf{NP}$, and others where $\mathsf{P} \neq \mathsf{NP}$. For that reason, any proof of $\mathsf{P} \neq \mathsf{NP}$ will need to "notice," at some point,

that there are no oracles in "our" world: it will have to use techniques that *fail* relative to certain oracles.

Theorem 42 (Baker-Gill-Solovay [36]) *There exists an oracle A such that $\mathsf{P}^A = \mathsf{NP}^A$, and another oracle B such that $\mathsf{P}^B \neq \mathsf{NP}^B$.*

Proof Sketch. To make $\mathsf{P}^A = \mathsf{NP}^A$, we can just let A be any PSPACE-complete language. Then it's not hard to see that $\mathsf{P}^A = \mathsf{NP}^A = \mathsf{PSPACE}$.

To make $\mathsf{P}^B \neq \mathsf{NP}^B$, we can (for example) let B be a random oracle, as observed by Bennett and Gill [44]. We can then, for example, define

$$L = \{0^n : \text{the first } 2^n \text{ bits of } B \text{ contain a run of } n \text{ consecutive 1's}\}.$$

Clearly $L \in \mathsf{NP}^B$. By contrast, one can easily show that $L \notin \mathsf{P}^B$ with probability 1 over B: in this case, there *really is* nothing for a deterministic Turing machine to do but brute-force search, requiring exponentially many queries to the B oracle. ■

We also have the following somewhat harder result.

Theorem 43 (Wilson [267]) *There exists an oracle A such that $\mathsf{NEXP}^A \subset \mathsf{P}^A/\mathsf{poly}$, and such that every language in NP^A has linear-sized circuits with A-oracle gates (that is, gates that query A).*

In other words, any proof even of $\mathsf{NEXP} \not\subset \mathsf{P/poly}$—that is, of a circuit lower bound just "slightly" beyond those that have already been proven—will require non-relativizing techniques. One can likewise show that non-relativizing techniques will be needed to make real progress on many of the other open problems of complexity theory (such as proving $\mathsf{P} = \mathsf{BPP}$).

If relativization seems too banal, the way to appreciate it is to try to invent techniques, for proving inclusions or separations among complexity classes, that *fail* to relativize. It's harder than it sounds! A partial explanation for this was given by Arora, Impagliazzo, and Vazirani [29], who reinterpreted the relativization barrier in logical terms. From their perspective, a relativizing proof is simply any proof that "knows" about complexity classes, only through axioms that assert the classes' closure properties, as well as languages that the classes *do* contain. (For example, P contains the empty language; if L_1 and L_2 are both in P, then so are Boolean combinations like $\overline{L_1}$ and $L_1 \cap L_2$.) These axioms can be shown to imply statements such as $\mathsf{P} \neq \mathsf{EXP}$. But other statements, like $\mathsf{P} \neq \mathsf{NP}$, can be shown to be independent of the axioms, by constructing models of the axioms where those statements are false. One constructs those models by using oracles to "force in" additional languages—such as PSPACE-complete languages, if one wants a world where $\mathsf{P} = \mathsf{NP}$—which the axioms might not *require* to be contained in complexity classes like P and NP, but which they don't prohibit from being contained, either. The conclusion is that any proof of $\mathsf{P} \neq \mathsf{NP}$ will need to appeal to deeper properties of the classes P and NP, properties that don't follow from these closure axioms.

6.2 Combinatorial Lower Bounds

Partly because of the relativization barrier, in the 1980s attention shifted to combinatorial approaches: that is, approaches where one tries to prove superpolynomial lower bounds on the number of operations of *some* kind needed to do *something*, by actually "rolling up one's sleeves" and

delving into the messy details of what the operations do (rather than making abstract diagonalization arguments). These combinatorial approaches enjoyed some spectacular successes, some of which seemed at the time like they were within striking distance of proving $\mathsf{P} \neq \mathsf{NP}$. Let's see some examples.

6.2.1 Proof Complexity

Suppose we're given a 3SAT formula φ, and we want to prove that φ has no satisfying assignments. One natural approach to this is called *resolution*: we repeatedly pick two clauses of φ, and then "resolve" the clauses (or "smash them together") to derive a new clause that logically follows from the first two. This is most useful when one of the clauses contains a non-negated literal x, and the other contains the corresponding negated literal $\overline{x}$. For example, from the clauses $(x \vee y)$ and $(\overline{x} \vee z)$, it's easy to see that we can derive $(y \vee z)$. The new derived clause can then be added to the list of clauses, and used as an input to future resolution steps.

Now, if we ever derive the empty clause ( )—say, by smashing together (x) and $(\overline{x})$—then we can conclude that our original 3SAT formula φ must have been unsatisfiable. For in that case, φ entails a clause that's not satisfied by *any* setting of variables. Another way to say this is that resolution is a *sound* proof system. By doing induction on the number of variables in φ, it's not hard to show that resolution is also *complete*:

Proposition 44 *Resolution is a complete proof system for the unsatisfiability of* kSAT. *In other words, given any unsatisfiable* kSAT *formula* φ*, there exists some sequence of resolution steps that produces the empty clause.*

So the key question about resolution is just *how many* resolution steps are needed to derive the empty clause, starting from an unsatisfiable formula φ. If that number could be upper-bounded by a polynomial in the size of φ, it would follow that $\mathsf{NP} = \mathsf{coNP}$. If, moreover, an appropriate sequence of resolutions could actually be *found* in polynomial time, it would follow that $\mathsf{P} = \mathsf{NP}$.

On the other hand, when we prove completeness by induction on the number of variables n, the only upper bound we get on the number of resolution steps is 2^n. And indeed, in 1985, Haken proved the following celebrated result.

Theorem 45 (Haken [111]) *There exist* kSAT *formulas, involving* $n^{O(1)}$ *variables and clauses, for which any resolution proof of unsatisfiability requires at least* $2^{\Omega(n)}$ *resolution steps. An example is a* kSAT *formula that explicitly encodes the "*n^{th} *Pigeonhole Principle": that is, the statement that there's no way to assign* $n+1$ *pigeons to* n *holes, without assigning two or more pigeons to the same hole.*

Haken's proof formalized the intuition that any resolution proof of the Pigeonhole Principle will ultimately be stuck "reasoning locally": "let's see, if I put this pigeon there, and that one there ... darn, it *still* doesn't work!" Such a proof has no ability to engage in higher-level reasoning about the total *number* of pigeons.[38]

[38] Haken's proof has since been substantially simplified. Readers interested in a "modern" version should consult Ben-Sasson and Wigderson [42], who split the argument into two easy steps: first, "short proofs are narrow"—that is, any short resolution proof can be converted into another such proof that only contains clauses with few literals—and second, any resolution refutation of a kSAT instance encoding (for example) the pigeonhole principle must be "wide," because of the instance's graph-expansion properties.

Since Haken's breakthrough, there have been many other exponential lower bounds on the sizes of unsatisfiability proofs, typically for proof systems that generalize resolution in some way (see Beame and Pitassi [38] for a good survey). These, in turn, often let us prove exponential lower bounds on the running times of certain kinds of algorithms. For example, there's a widely-used class of kSAT algorithms called DPLL (Davis-Putnam-Logemann-Loveland) algorithms [79], which are based on pruning the search tree of possible satisfying assignments. DPLL algorithms have the property that, if one looks at the search tree of their execution on an unsatisfiable kSAT formula φ, one can *read off* a resolution proof that φ is unsatisfiable. From that fact, together with Theorem 45, it follows that there exist kSAT formulas (for example, the Pigeonhole Principle formulas) for which any DPLL algorithm requires exponential time.

In principle, if one could prove superpolynomial lower bounds for *arbitrary* proof systems (constrained only by the proofs being checkable in polynomial time), one would get $\mathsf{P} \neq \mathsf{NP}$, and even $\mathsf{NP} \neq \mathsf{coNP}$! However, perhaps this motivates turning our attention to lower bounds on circuit size, which tend to be somewhat easier than the analogous proof complexity lower bounds, and which—if generalized to arbitrary Boolean circuits—would "merely" imply $\mathsf{P} \neq \mathsf{NP}$ and $\mathsf{NP} \not\subset \mathsf{P/poly}$, rather than $\mathsf{NP} \neq \mathsf{coNP}$.

6.2.2 Monotone Circuit Lower Bounds

Recall, from Section 5.2, that if we could merely prove that any family of *Boolean circuits* to solve some NP problem required a superpolynomial number of AND, OR, and NOT gates, then that would imply $\mathsf{P} \neq \mathsf{NP}$, and even the stronger result $\mathsf{NP} \not\subset \mathsf{P/poly}$ (that is, NP-complete problems are not efficiently solvable by nonuniform algorithms).

Now, some NP-complete languages L have the interesting property of being *monotone*: that is, changing an input bit from 0 to 1 can change the answer from $x \notin L$ to $x \in L$, but never from $x \in L$ to $x \notin L$. An example is the CLIQUE language: say, the set of all encodings of n-vertex graphs G, as adjacency matrices of 0s and 1s, such that G contains a clique on at least $\sqrt{n}$ vertices. It's not hard to see that we can decide any such language using a *monotone circuit*: that is, a Boolean circuit of AND and OR gates only, no NOT gates. For the CLIQUE language, for example, a circuit could simply consist of an OR of $\binom{n}{\sqrt{n}}$ ANDs, one for each possible clique. It thus becomes interesting to ask what are the *smallest* monotone circuits for monotone NP-complete languages.

In 1985, Alexander Razborov, then a graduate student, astonished the complexity theory world with the following result.

Theorem 46 (Razborov [206]) *Any monotone circuit for* CLIQUE *requires at least* $n^{\Omega(\log n)}$ *gates.*

Subsequently, Alon and Boppana [26] improved this, to show that any monotone circuit to detect a clique of size $\sim (n/\log n)^{2/3}$ must have size $\exp\left(\Omega\left((n/\log n)^{1/3}\right)\right)$. I won't go into the proof of Theorem 46 here, but it uses beautiful combinatorial techniques, including (in modern versions) the Erdös-Rado sunflower lemma.

The significance of Theorem 46 is this: if we could now merely prove that *any circuit for a monotone language can be made into a monotone circuit without much increasing its size*, then we'd immediately get $\mathsf{P} \neq \mathsf{NP}$ and even $\mathsf{NP} \not\subset \mathsf{P/poly}$. Indeed, this was considered a potentially-viable approach to proving $\mathsf{P} \neq \mathsf{NP}$ for some months. Alas, the approach turned out to be a dead end, because of a result first shown by Razborov himself (and then improved by Tardos):

Theorem 47 (Razborov [207], Tardos [241]) *There are monotone languages in* P *that require exponentially-large monotone circuits. An example is the* MATCHING *language, consisting of all adjacency-matrix encodings of n-vertex graphs that admit a matching on at least $\sim (n/\log n)^{2/3}$ vertices. This language requires monotone circuits of size $\exp\left(\Omega\left((n/\log n)^{1/3}\right)\right)$.*

Thus, while Theorem 46 stands as a striking example of the power of combinatorics to prove circuit lower bounds, ultimately it tells us not about the hardness of NP-complete problems, but only about the weakness of monotone circuits. Theorem 47 implies that, even if we're trying to compute a monotone Boolean function (such as the MATCHING function), allowing ourselves the non-monotone NOT gate can yield an exponential reduction in circuit size. But Razborov's techniques break down completely as soon as a few NOT gates are available.[39]

I should also mention lower bounds on monotone *depth*. In the STCON (s,t-connectivity) problem, we're given as input the adjacency matrix of an undirected graph, and asked whether or not there's a path between two designated vertices s and t. By using their connection between circuit depth and communication complexity (see Section 6), Karchmer and Wigderson [133] were able to prove that any monotone circuit for STCON requires $\Omega\left(\log^2 n\right)$ depth—and as a consequence, that any monotone *formula* for STCON requires $n^{\Omega(\log n)}$ size. Since STCON is known to have monotone circuits of polynomial size, this implies in particular that monotone formula size and monotone circuit size are not polynomially related.

6.2.3 Small-Depth Circuits and the Random Restriction Method

Besides restricting the allowed gates (say, to AND and OR only), there's a second natural way to "hobble" a circuit, and thereby potentially make it easier to prove lower bounds on circuit size. Namely, we can restrict the circuit's *depth*, the number of layers of gates between input and output. If the allowed gates all have a fanin of 1 or 2 (that is, they all take only 1 or 2 input bits), then clearly any circuit that depends nontrivially on all n of the input bits must have depth at least $\log_2 n$. On the other hand, if we allow gates of *unbounded fanin*—for example, ANDs or XORs or MAJORITYs on unlimited numbers of inputs—then it makes sense to ask what can be computed even by circuits of *constant* depth. Constant-depth circuits are very closely related to *neural networks*, which also consist of a small number of layers of "logic gates" (i.e., the neurons), with each neuron allowed to have very large "fanin"—i.e., to accept input from many or all of the neurons in the previous layer.

If we don't also restrict the number of gates or neurons, then it turns out that *every* function can be computed in small depth:

Proposition 48 *Every Boolean function $f : \{0,1\}^n \rightarrow \{0,1\}$ can be computed by an unbounded-fanin, depth-3 circuit of size $O\left(n2^n\right)$: namely, by an OR of ANDs of input bits and their negations.*

Proof. We simply need to check whether the input, $x \in \{0,1\}^n$, is one of the z's such that

[39] Note that, if we encode the input string using the so-called *dual-rail representation*—in which every 0 is represented by the 2-bit string 01, and every 1 by 10—then the monotone circuit complexities of CLIQUE, MATCHING, and so on *do* become essentially equivalent to their non-monotone circuit complexities, since we can push all the NOT gates to the bottom layer of the circuit using de Morgan's laws, and then eliminate the NOT gates using the dual-rail encoding. Unfortunately, Razborov's lower bound techniques also break down under dual-rail encoding.

$f(z) = 1$:

$$f(x) = \bigvee_{z=z_1 \cdots z_n \,:\, f(z)=1} \left(\left(\bigwedge_{i \in \{1,\ldots,n\} \,:\, z_i=1} x_i \right) \wedge \left(\bigwedge_{i \in \{1,\ldots,n\} \,:\, z_i=0} \overline{x}_i \right) \right).$$

■

Similarly, in typical neural network models, every Boolean function can be computed by a network with $\sim 2^n$ neurons arranged into just 2 layers.

So the interesting question is what happens if we restrict both the depth *and* the number of gates or neurons. More formally, let AC^0 be the class of languages $L \subseteq \{0,1\}^*$ for which there exists a family of circuits $\{C_n\}_{n \geq 1}$, one for each input size n, such that:

(1) $C_n(x)$ outputs 1 if $x \in L$ and 0 if $x \notin L$, for all n and $x \in \{0,1\}^n$.

(2) Each C_n consists of unbounded-fanin AND and OR gates, as well as NOT gates.

(3) There is a polynomial p such that each C_n has at most $p(n)$ gates.

(4) There is a constant d such that each C_n has depth at most d.

Clearly AC^0 is a subclass of $\mathsf{P/poly}$; indeed we recover $\mathsf{P/poly}$ by omitting condition (4). Now, one of the major triumphs of complexity theory in the 1980s was to understand AC^0, as we still only dream of understanding $\mathsf{P/poly}$. It's not just that we know $\mathsf{NP} \not\subset \mathsf{AC}^0$; rather, it's that we know in detail which problems are and aren't in AC^0 (even problems within P), and exactly how many gates are needed for each given depth d. As the most famous example, let PARITY be the language consisting of all strings with an odd number of '1' bits. Then:

Theorem 49 (Ajtai [17], Furst-Saxe-Sipser [94]) PARITY *is not in* AC^0.

While the original lower bounds on the size of AC^0 circuits for PARITY were only slightly superpolynomial, Theorem 49 was subsequently improved by Yao [269] and then by Håstad [113], the latter of whom gave an essentially optimal result: namely, any AC^0 circuit for PARITY of depth d requires at least $2^{\Omega\left(n^{1/(d-1)}\right)}$ gates.

The first proofs of Theorem 49 used what's called the *method of random restrictions*. In this method, we assume by contradiction that we have a size-s, depth-d, unbounded-fanin circuit C for our Boolean function—say, the PARITY function. We then randomly fix most of the input bits to 0 or 1, while leaving a few input bits unfixed. What we hope to find is that the random restriction "kills off" an entire layer of gates—because any AND gate that takes even one constant 0 bit as input can be replaced by the constant 0 function, and likewise, any OR gate that takes even one 1 bit as input can be replaced by the constant 1 function. Thus, any AND or OR gate with a large fanin is extremely likely to be killed off; gates with small fanin might not be killed off, but can be left around to be dealt with later. We then repeat this procedure, randomly restricting most of the remaining unfixed bits, in order to kill off the next higher layer of AND and OR gates, and so on through all d layers. By the time we're done, we've reduced C to a shadow of its former self: specifically, to a circuit that depends on only a constant number of input bits. Meanwhile, even though only a tiny fraction of the input bits (say, $n^{1/d}$ of them) remain unfixed, we still have a nontrivial Boolean function on those bits: indeed, it's easy to see that any restriction of the PARITY

function to a subset S of bits will either be PARITY itself, or else NOT(PARITY). But a circuit of constant size clearly can't compute a Boolean function that depends on $\sim n^{1/d}$ input bits. This yields our desired contradiction.

At a high level, there were three ingredients needed for the random restriction method to work. First, the circuit needed to built out of AND and OR gates, which are likely to get killed off by random restrictions. The method *wouldn't* have worked if the circuit contained unbounded-fanin MAJORITY gates (as a neural network does), or even unbounded-fanin XOR gates. Second, it was crucial that the circuit depth d was small, since we needed to shrink the number of unfixed input variables by a large factor d times, and then still have unfixed variables left over. It turns out that random restriction arguments can yield *some* lower bound whenever $d = o\left(\frac{\log n}{\log \log n}\right)$, but not beyond that. Third, we needed to consider a function, such as PARITY, that remains nontrivial even after the overwhelming majority of input bits are randomly fixed to 0 or 1. The method wouldn't have worked, for example, for the n-bit AND function (which is unsurprising, since the AND function *does* have a depth-1 circuit, consisting of a single AND gate!).

The original proofs for PARITY $\notin \mathsf{AC}^0$ have been generalized and improved on in many ways. For example, Linial, Mansour, and Nisan [160] examined the weakness of AC^0 circuits from a different angle: "turning lemons into lemonade," they gave a quasipolynomial-time algorithm to *learn* arbitrary AC^0 circuits with respect to the uniform distribution over inputs.[40] Also, proving a conjecture put forward by Linial and Nisan [161] (and independently Babai), Braverman [55] showed that AC^0 circuits can't distinguish the outputs of a wide range of pseudorandom generators from truly random strings.

Meanwhile, Håstad [113] showed that for every d, there are functions computable by AC^0 circuits of depth d that require exponentially many gates for AC^0 circuits of depth $d-1$. This implies that there exists an oracle relative to which PH is infinite (that is, all its levels are distinct). Improving that result, Rossman, Servedio, and Tan [212] very recently showed that the same functions Håstad had considered require exponentially many gates even to *approximate* using AC^0 circuits of depth $d-1$. This implies that PH is infinite relative to a *random* oracle with probability 1, which resolved a thirty-year-old open problem.

The random restriction method has also had other applications in complexity theory, besides to AC^0. Most notably, it's been used to prove polynomial lower bounds on *formula size*. The story of formula-size lower bounds starts in 1961 with Subbotovskaya [237], who used random restrictions to show that the n-bit PARITY function requires formulas of size $\Omega\left(n^{1.5}\right)$. Later Khrapchenko [141] improved this to $\Omega\left(n^2\right)$, which is tight.[41] Next, in 1987, Andreev [27] constructed a different Boolean function in P that could be shown, again using random restrictions, to require formulas of size $n^{2.5-o(1)}$. This was subsequently improved to $n^{2.55-o(1)}$ by Impagliazzo and Nisan [125], to $n^{2.63-o(1)}$ by Paterson and Zwick [194], and finally to $n^{3-o(1)}$ by Håstad [114] and to $\Omega\left(\frac{n^3}{(\log n)^2(\log \log n)^3}\right)$ by Tal [240]. Unfortunately, the random restriction method seems fundamentally incapable of going beyond $\Omega\left(n^3\right)$. On the other hand, for Boolean circuits rather than formulas, we still have no lower bound better than *linear* for any function in P (or for that

[40] By a "learning algorithm," here we mean an algorithm that takes as input uniformly-random samples $x_1, \ldots, x_k \in \{0,1\}^n$, as well as $f(x_1), \ldots, f(x_k)$, where f is some unknown function in AC^0, and that with high probability over the choice of $x_1, \ldots, x_k$, outputs a hypothesis h such that $\Pr_{x \in \{0,1\}^n}[h(x) = f(x)]$ is close to 1.

[41] Assume for simplicity that n is a power of 2. Then $x_1 \oplus \cdots \oplus x_n$ can be written as $y \oplus z$, where $y := x_1 \oplus \cdots \oplus x_{n/2}$ and $z := x_{n/2+1} \oplus \cdots \oplus x_n$. This in turn can be written as $(y \wedge \overline{z}) \vee (\overline{y} \wedge z)$. Expanding recursively now yields a size-n^2 formula for PARITY, made of AND, OR, and NOT gates.

matter, in NP)!

6.2.4 Small-Depth Circuits and the Polynomial Method

For our purposes, the most important extension of Theorem 49 was achieved by Smolensky [229] and Razborov [208] in 1987. Let $\mathsf{AC}^0[m]$ be the class of languages decidable by a family of constant-depth, polynomial-size, unbounded-fanin circuits with AND, OR, NOT, and MOD-m gates (which output 1 if their number of '1' input bits is divisible by m, and 0 otherwise). Adding in MOD-m gates seems like a natural extension of AC^0: for example, if $m = 2$, then we're just adding PARITY, one of the most basic functions not in AC^0.

However, Smolensky and Razborov extended the class of circuits for which lower bounds can be proven from AC^0 to $\mathsf{AC}^0[p]$, whenever p is prime.

Theorem 50 (Smolensky [229], Razborov [208]) *Let p and q be distinct primes. Then MOD_q, the set of all strings with Hamming weight divisible by q, is not in $\mathsf{AC}^0[p]$. Indeed, any $\mathsf{AC}^0[p]$ circuit for MOD_q of depth d requires $2^{\Omega\left(n^{1/2d}\right)}$ gates. As a corollary, the* MAJORITY *function is also not in $\mathsf{AC}^0[p]$, and also requires $2^{\Omega\left(n^{1/2d}\right)}$ gates to compute using $\mathsf{AC}^0[p]$ circuits of depth d.*

It's not hard to show that $\mathsf{AC}^0[p] = \mathsf{AC}^0\left[p^k\right]$ for any $k \geq 1$, and thus, one also gets lower bounds against $\mathsf{AC}^0[m]$, whenever m is a prime power.

The proof of Theorem 50 uses the so-called *polynomial method*. Here one argues that, if a function f can be computed by a constant-depth circuit with AND, OR, NOT, and MOD-p gates, then f can also be approximated by a low-degree polynomial over the finite field $\mathbb{F}_p$. One then shows that a function of interest, such as the MOD_q function (for $q \neq p$), *can't* be approximated by any such low-degree polynomial. This provides the desired contradiction.

The polynomial method is famously specific in scope: it's still not known how to prove results like Theorem 50 even for $\mathsf{AC}^0[m]$ circuits, where m is not a prime power.[42] The reason why it breaks down there is simply that there are no finite fields of non-prime-power order. And thus, to be concrete, it's still open whether the n-bit MAJORITY function has a constant-depth, polynomial-size, unbounded-fanin circuit consisting of AND, OR, NOT, and MOD-6 gates, or even entirely of MOD-6 gates!

Stepping back, it's interesting to ask whether the constant-depth circuit lower bounds evade the relativization barrier explained in Section 6.1.2. There's some disagreement about whether it's even sensible to feed oracles to tiny complexity classes such as AC^0 (see Allender and Gore [23] for example). However, to whatever extent it *is* sensible, the answer is that these lower bounds do evade relativization. For example, if by $\left(\mathsf{AC}^0\right)^A$, we mean AC^0 extended by "oracle gates" that query A, then it's easy to construct an A such that $\left(\mathsf{AC}^0\right)^A = \mathsf{P}^A$: for example, any A that is P-complete under AC^0-reductions will work. On the other hand, we know from Theorem 49 that $\mathsf{AC}^0 \neq \mathsf{P}$ in the "real," unrelativized world.

6.2.5 The Natural Proofs Barrier

Despite the weakness of AC^0 and $\mathsf{AC}^0[p]$ circuits, the progress on lower bounds for them suggested what seemed to many researchers like a plausible path to proving $\mathsf{NP} \not\subset \mathsf{P/poly}$, and hence $\mathsf{P} \neq \mathsf{NP}$.

[42] Williams's $\mathsf{NEXP} \not\subset \mathsf{ACC}$ breakthrough [265], to be discussed in Section 6.4.2, could be seen as the first successful use of the "polynomial method" to prove a lower bound against $\mathsf{AC}^0[m]$ circuits—though in that case, the lower bound applies only to NEXP-complete problems, and polynomials are only one ingredient in the proof among many.

That path is simply to generalize the random restriction and polynomial methods further and further, to get lower bounds for more and more powerful classes of circuits. The first step, of course, would be to generalize the polynomial method to handle $\mathsf{AC}^0[m]$ circuits, where m is not a prime power. Then one could handle what are called TC^0 circuits: that is, constant-depth, polynomial-size, unbounded-fanin circuits with MAJORITY gates (or, as in a neural network, *threshold gates*, which output 1 if a certain weighted affine combination of the input bits exceeds 0, and 0 otherwise). Next, one could aim for polynomial-size circuits of logarithmic depth: that is, the class NC^1. Finally, one could push all the way to polynomial-depth circuits: that is, the class $\mathsf{P/poly}$.

Unfortunately, we now know that this path hits a profound barrier at TC^0, if not earlier—a barrier that explains why the random restriction and polynomial methods haven't taken us further toward a proof of $\mathsf{P} \neq \mathsf{NP}$. Apparently this barrier was known to Michael Sipser (and perhaps a few others) in the 1980s, but it was first articulated in print in 1993 by Razborov and Rudich [210], who called it the *natural proofs barrier*.

The basic insight is that combinatorial techniques, such as the method of random restrictions, do more than advertised: in some sense, they do too much for their own good. In particular, not only do they let us show that certain specific functions, like PARITY, are hard for AC^0; they even let us certify that a *random* function is hard for AC^0. Indeed, such techniques give rise to an *algorithm*, which takes as input the truth table of a Boolean function $f : \{0,1\}^n \rightarrow \{0,1\}$, and which has the following two properties.

(1) **"Constructivity."** The algorithm runs in time polynomial in the size of f's truth table (that is, polynomial in 2^n).

(2) **"Largeness."** If f is chosen uniformly at random, then with probability at least $1/n^{O(1)}$ over f, the algorithm certifies that f is hard (i.e., that f is not in some circuit class $\mathcal{C}$, such as AC^0 in the case of the random restriction method).

If a lower bound proof gives rise to an algorithm satisfying (1) and (2), then Razborov and Rudich call it a *natural proof*. In many cases, it's not entirely obvious that a lower bound proof is natural, but with some work one can show that it is. To illustrate, in the case of the random restriction method, the algorithm could check that f has a large fraction of its Fourier mass on high-degree Fourier coefficients, or that f has high "average sensitivity" (that is, if x and y are random inputs that differ only in a single bit, then with high probability $f(x) \neq f(y)$). These tests have the following three properties:

- They're easy to perform, in time polynomial in the truth table size 2^n.
- A random function f will pass these tests with overwhelming probability (that is, such an f will "look like PARITY" in the relevant respects).
- The results of Linial, Mansour, and Nisan [160] show that any f that passes these tests remains nontrivial under most random restrictions, and for that reason, can't be in AC^0.

But now, twisting the knife, Razborov and Rudich point out that any natural lower bound proof against a powerful enough complexity class would be self-defeating, in that *it would yield an efficient algorithm to solve some of the same problems that we'd set out to prove were hard.* More concretely,

suppose we have a natural lower bound proof against the circuit class $\mathcal{C}$. Then by definition, we also have an efficient algorithm A that, given a random Boolean function $f : \{0,1\}^n \rightarrow \{0,1\}$, certifies that $f \notin \mathcal{C}$ with at least $1/n^{O(1)}$ probability over f. But this means that $\mathcal{C}$ cannot contain very strong families of *pseudorandom functions*: namely, functions $f : \{0,1\}^n \rightarrow \{0,1\}$ that are indistinguishable from "truly" random functions, even by algorithms that can examine their entire truth tables and use time polynomial in 2^n.

Why not? Because A can *never* certify $f \notin \mathcal{C}$ if f is a pseudorandom function, computable in $\mathcal{C}$. But A certifies $f \notin \mathcal{C}$ with $1/n^{O(1)}$ probability over a truly random f. Thus, A serves to distinguish random from pseudorandom functions with non-negligible[43] bias—so the latter were never really pseudorandom at all.

To recap, we've shown that, if there's any natural proof that any function is not in $\mathcal{C}$, then all Boolean functions computable in $\mathcal{C}$ can be distinguished from random functions by $2^{O(n)}$-time algorithms. That might not sound so impressive, since $2^{O(n)}$ is a lot of time. But a key observation is that, for most of the circuit classes $\mathcal{C}$ that we care about, there are families of pseudorandom functions $\{f_s\}_s$ on n bits that are conjectured to require $2^{p(n)}$ time to distinguish from truly random functions, where $p(n)$ is as large a polynomial as we like (related to the length of the random "seed" s). It follows from results of Naor and Reingold [187] that in TC^0 (constant-depth, polynomial-size threshold circuits), there are functions that can't be distinguished from random functions in $2^{O(n)}$ time, *unless* the factoring and discrete logarithm problems are solvable in $O\left(2^{n^\varepsilon}\right)$ time for every $\varepsilon > 0$. (For comparison, the best *known* algorithms for these problems take roughly $2^{n^{1/3}}$ time.) Likewise, Banerjee et al. [37] showed that in TC^0, there are functions that can't be distinguished from random in $2^{O(n)}$ time, unless noisy systems of linear equations can be solved in $O\left(2^{n^\varepsilon}\right)$ time for every $\varepsilon > 0$.

It's worth pausing to let the irony sink in. Razborov and Rudich are pointing out that, as we showed certain problems (factoring and discrete logarithm) to be harder and harder via a natural proof, we'd simultaneously show those same problems to be easier and easier! Indeed, any natural proof showing that these problems took *at least* $t(n)$ time, would also show that they took *at most* roughly $2^{t^{-1}(n)}$ time. As a result, no natural proof could possibly show these problems take more than half-exponential time: that is, time $t(n)$ such that $t(t(n))$ grows exponentially.

Here, perhaps, we're finally face-to-face with a central conceptual difficulty of the $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ question: namely, we're trying to prove that certain functions are hard, but the problem of deciding whether a function is hard is *itself* hard, according to the very sorts of conjectures that we're trying to prove.[44]

Of course, the natural proofs barrier didn't prevent complexity theorists from proving strong lower bounds against AC^0. But the result of Linial, Mansour, and Nisan [160] can be interpreted as saying that this is *because* AC^0 is not yet powerful enough to express pseudorandom functions. When we move just slightly higher, to TC^0 (constant-depth threshold circuits), we *do* have pseudorandom functions under plausible hardness assumptions, and—not at all coincidentally, according to Razborov and Rudich—we no longer have strong circuit lower bounds. In that sense, natural proofs explains almost precisely why the progress toward proving $\mathsf{P} \neq \mathsf{NP}$ via circuit complexity stalled where it did. The one complication in the story is the $\mathsf{AC}^0[m]$ classes, for which we don't

[43] In theoretical computer science, the term *non-negligible* means lower-bounded by $1/n^{O(1)}$.

[44] Technically, the problem of distinguishing random from pseudorandom functions is equivalent to the problem of inverting one-way functions, which is not *quite* as strong as solving NP-complete problems in polynomial time—only solving average-case NP-complete problems with planted solutions. For more see Section 5.3.

yet have strong lower bounds (though see Section 6.4), but *also* don't have pseudorandom function candidates. For those classes, it's still possible that natural proofs could succeed.

As Razborov and Rudich themselves stressed, the take-home message is *not* that we should give up on proving $\mathsf{P} \neq \mathsf{NP}$. In fact, since the beginning of complexity theory, we've had at least one technique that easily evades the natural proofs barrier: namely, diagonalization (the technique used to prove $\mathsf{P} \neq \mathsf{EXP}$; see Section 6.1)! The reason why diagonalization evades the barrier is that it zeroes in on a specific property of the function f being lower-bounded—namely, the fact that f is EXP-complete, and thus able to simulate all P machines—and thereby avoids the trap of arguing that "f is hard because it looks like a random function." Of course, diagonalization is subject to the relativization barrier (see Section 6.1.2), so the question still stands of how to evade relativization and natural proofs simultaneously; we'll return to that question in Section 6.3.

More broadly, there are many cases in mathematics where we can prove that some object O of interest to us has a property P, even though we have no hope of finding a general polynomial-time algorithm to decide whether *any* given object has property P, or even to certify a large fraction of objects as having property P. In such cases, often we prove that O has property P by exploiting special symmetries in O—symmetries that have little to do with why O has property P, but everything to do with why we can *prove* it has the property. As an example, a random graph is an *expander graph* (that is, a graph on which a random walk mixes rapidly) with overwhelming probability. But since the general problem of deciding whether a graph is an expander is NP-hard, if we want a *specific* graph G that's provably an expander, typically we need to construct G with a large amount of symmetry: for example, by taking it to be the Cayley graph of a finite group. Similarly, even though we expect that there's no general efficient algorithm to decide if a Boolean function f is hard,[45] given as input f's truth table, we might be able to prove that certain *specific* f's (for example, NP- or $\#\mathsf{P}$-complete ones) are hard by exploiting their symmetries. Geometric Complexity Theory (see Section 6.6) is the best-known development of that particular hope for escaping the natural proofs barrier.

But GCT is not the only way to use symmetry to evade natural proofs. As a vastly smaller example, I [2, Appendix 10] proved an exponential lower bound on the so-called *manifestly orthogonal formula size* of a function $f : \{0,1\}^n \rightarrow \{0,1\}$ that outputs 1 if the input x is a codeword of a linear error-correcting code, and 0 otherwise. Here a *manifestly orthogonal formula* is a formula over $x_1, \ldots, x_n, \overline{x}_1, \ldots, \overline{x}_n$ consisting of OR and AND gates, where every OR must be of two subformulas over the same set of variables, and every AND must be of two subformulas over disjoint sets of variables. My lower bound wasn't especially difficult, but what's notable about it is that it took crucial advantage of a *symmetry* of linear error-correcting codes: namely, the fact that any such code can be recursively decomposed as a disjoint union of Cartesian products of smaller linear error-correcting codes. My proof thus gives no apparent insight into how to certify that a *random* Boolean function has manifestly orthogonal formula size $\exp(n)$, and possibly evades the natural

[45] The problem, given as input the truth table of a Boolean function $f : \{0,1\}^n \rightarrow \{0,1\}$, of computing or approximating the circuit complexity of f is called the Minimum Circuit Size Problem (MCSP). It's a longstanding open problem whether or not MCSP is NP-hard; at any rate, there are major obstructions to *proving* it NP-hard with existing techniques (see Kabanets and Cai [129] and Murray and Williams [186]). On the other hand, MCSP can't be in P (or BPP) unless there are no cryptographically-secure pseudorandom generators. At any rate, what's relevant to natural proofs is just whether there's an efficient algorithm to *certify a large fraction* of Boolean functions as being hard: that is, to output "f is hard" for a $1/\operatorname{poly}(n)$ fraction of Boolean functions $f : \{0,1\}^n \rightarrow \{0,1\}$ that require large circuits, and for no Boolean functions that have small circuits. This is a weaker requirement than solving MCSP.

proofs barrier (if there *is* such a barrier in the first place for manifestly orthogonal formulas).

Another proposal for how to evade the natural proofs barrier comes from a beautiful 2010 paper by Allender and Koucký [25] (see also Allender's survey [21]). These authors show that, if one wanted to prove that certain specific NC^1 problems were not in TC^0, thereby establishing the breakthrough separation $\mathsf{TC}^0 \neq \mathsf{NC}^1$, it would suffice to show that those problems had no TC^0 circuits of size $n^{1+\varepsilon}$, for any constant $\varepsilon > 0$. To achieve this striking "bootstrap," from an $n^{1+\varepsilon}$ lower bound to a superpolynomial one, Allender and Koucký exploit the *self-reducibility* of the NC^1 problems in question: the fact that they can be reduced to smaller instances of themselves. Crucially, this self-reducibility would *not* hold for a random function. For this reason, the proposed lower bound method has at least the potential to evade the natural proofs barrier. Indeed, it's not even totally implausible that a natural proof could yield an $n^{1+\varepsilon}$ lower bound for TC^0 circuits, with the known bootstrapping from $n^{1+\varepsilon}$ to superpolynomial the only non-natural part of the argument.[46]

I can't resist mentioning a final idea about how to evade natural proofs. In a 2014 paper, Chapman and Williams [70] suggested proving circuit lower bounds for NP-complete problems like 3Sat, via arguments that would work only for circuits that are *self-certifying*: that is, that output satisfying assignments whenever they exist, which we know that a circuit solving NP-complete problems can always do by Proposition 4. Strikingly, they then showed that if $\mathsf{NP} \not\subset \mathsf{P/poly}$ is true at all, then it has a proof that's "natural" in their modified sense: that is, a proof that yields an efficient algorithm to certify that a $1/n^{O(1)}$ fraction of all Boolean functions $f : \{0,1\}^n \rightarrow \{0,1\}$ *either don't have polynomial-size circuits or else aren't self-certifying*. Thus, if we could just figure out how to exploit NP-complete problems' property of self-certification, that would already be enough to evade natural proofs.

6.3 Arithmetization

In the previous sections, we saw that there are logic-based techniques (like diagonalization) that suffice to prove $\mathsf{P} \neq \mathsf{EXP}$ and $\mathsf{NEXP}^{\mathsf{NP}} \not\subset \mathsf{P/poly}$, and that evade the natural proofs barrier, but that are blocked from proving $\mathsf{P} \neq \mathsf{NP}$ by the relativization barrier. Meanwhile, there are combinatorial techniques (like random restrictions) that suffice to prove circuit lower bounds against AC^0 and $\mathsf{AC}^0[p]$, and that evade the relativization barrier, but that are blocked from proving lower bounds against $\mathsf{P/poly}$ (and hence, from proving $\mathsf{P} \neq \mathsf{NP}$) by the natural proofs barrier.

This raises a question: couldn't we simply *combine* techniques that evade relativization but not natural proofs, with techniques that evade natural proofs but not relativization, in order to evade both? As it turns out, we can.

6.3.1 IP = PSPACE

The story starts with a dramatic development in complexity theory around 1990, though not one that obviously bore on $\mathsf{P} \neq \mathsf{NP}$ or circuit lower bounds. In the 1980s, theoretical cryptographers became interested in so-called *interactive proof systems*, which are protocols where a

[46] Allender and Koucký's paper partly builds on 2003 work by Srinivasan [231], who showed that, to prove $\mathsf{P} \neq \mathsf{NP}$, one would "merely" need to show that any algorithm to compute weak approximations for the MaxClique problem takes $\Omega\left(n^{1+\varepsilon}\right)$ time, for some constant $\varepsilon > 0$. The way Srinivasan proved this striking statement was, again, by using a sort of self-reduciblity: he showed that, if there's a polynomial-time algorithm for MaxClique, then by running that algorithm on smaller graphs sampled from the original graph, one can solve approximate versions of MaxClique in $n^{1+o(1)}$ time.

computationally-unbounded but untrustworthy prover (traditionally named Merlin) tries to convince a skeptical polynomial-time verifier (traditionally named Arthur) that some mathematical statement is true, via a two-way conversation, in which Arthur can randomly generate challenges and then evaluate Merlin's answers to them.

More formally, let IP (Interactive Proof) be the class of all languages $L \subseteq \{0,1\}^*$ for which there exists a probabilistic polynomial-time algorithm for Arthur with the following properties. Arthur receives an input string $x \in \{0,1\}^n$ (which Merlin also knows), and then generates up to $n^{O(1)}$ challenges to send to Merlin. Each challenge is a string of up to $n^{O(1)}$ bits, and each can depend on x, on Arthur's internal random bits, *and* on Merlin's responses to the previous challenges. (We also allow Arthur, if he likes, to keep some random bits hidden, without sending them to Merlin—though surprisingly, this turns out not to make any difference [100].) We think of Merlin as trying his best to persuade Arthur that $x \in L$; at the end, Arthur decides whether to accept or reject Merlin's claim. We require that for all inputs x:

- If $x \in L$, then there's some strategy for Merlin (i.e., some function determining which message to send next, given x and the sequence of challenges so far[47]) that causes Arthur to accept with probability at least $2/3$ over his internal randomness.
- If $x \notin L$, then regardless of what strategy Merlin uses, Arthur rejects with probability at least $2/3$ over his internal randomness.

Clearly IP generalizes NP: indeed, we recover NP if we get rid of the interaction and randomness aspects, and just allow a single message from Merlin, which Arthur either accepts or rejects. In the other direction, it's not hard to show that $\mathsf{IP} \subseteq \mathsf{PSPACE}$.[48]

The question asked in the 1980s was: *does interaction help?* In other words, how much bigger is IP than NP*?* It was observed that IP contains at least a few languages that aren't known to be in NP, such as graph non-isomorphism. This is so because of a simple, famous, and elegant protocol [99]: given two n-vertex graphs G and H, Arthur can pick one of the two uniformly at random, randomly permute its vertices, then send the result to Merlin. He then challenges Merlin: *which graph did I start from, G or H?* If $G \ncong H$, then Merlin, being computationally unbounded, can easily answer this challenge by solving graph isomorphism. If, on the other hand, $G \cong H$, then Merlin sees the same distribution over graphs regardless of whether Arthur started from G or H, so he must guess wrongly with probability $1/2$.

Despite such protocols, the feeling in the late 1980s was that IP should be only a "slight" extension of NP. This feeling was buttressed by a result of Fortnow and Sipser [91], which said that there exists an oracle A such that $\mathsf{coNP}^A \not\subset \mathsf{IP}^A$, and hence, any interactive protocol even for coNP (e.g., for proving Boolean formulas unsatisfiable) would require non-relativizing techniques.

Yet in the teeth of that oracle result, Lund, Fortnow, Karloff, and Nisan [167] showed nevertheless that $\mathsf{coNP} \subseteq \mathsf{IP}$ "in the real world"—and not only that, but $\mathsf{P}^{\#\mathsf{P}} \subseteq \mathsf{IP}$. This was quickly improved by Shamir [220] to the following striking statement:

[47] We can assume without loss of generality that Merlin's strategy is deterministic, since Merlin is computationally unbounded, and any convex combination of strategies must contain a deterministic strategy that causes Arthur to accept with at least as great a probability as the convex combination does.

[48] This is so because a polynomial-space Turing machine can treat the entire interaction between Merlin and Arthur as a game, in which Merlin is trying to get Arthur to accept with the largest possible probability. The machine can then evaluate the exponentially large game tree using depth-first recursion.

Theorem 51 ([167, 220]) IP = PSPACE.

Theorem 51 means, for example, that if a computationally-unbounded alien came to Earth, it could not merely beat us in games of strategy like chess: rather, the alien could mathematically prove to us, via a short conversation and to statistical certainty, that it knew how to play *perfect* chess.[49] Theorem 51 has been hugely influential in complexity theory for several reasons, but one reason was that it illustrated, dramatically and indisputably, that the relativization barrier need not inhibit progress.

So how was this amazing result achieved, and why does the proof *fail* relative to certain oracles? The trick is what we now call *arithmetization*. This means that we take a Boolean formula or circuit—involving, for example, AND, OR, and NOT gates—and then reinterpret the Boolean gates as arithmetic operations over some larger finite field $\mathbb{F}_p$. More concretely, the Boolean AND ($x \wedge y$) becomes multiplication (xy), the Boolean NOT becomes the function $1-x$, and the Boolean OR ($x \vee y$) becomes $x+y-xy$. Note that if $x, y \in \{0,1\}$, then we recover the original Boolean operations. But the new operations make sense even if $x, y \notin \{0,1\}$, and they have the effect of lifting our Boolean formula or circuit to a multivariate polynomial over $\mathbb{F}_p$. Furthermore, the degree of the polynomial can be upper-bounded in terms of the size of the formula or circuit.

The advantage of this lifting is that polynomials, at least over large finite fields, have powerful error-correcting properties that are unavailable in the Boolean case. These properties ultimately derive from the Fundamental Theorem of Algebra: a nonzero, degree-d univariate polynomial has at most d roots. As a consequence, if $q, q' : \mathbb{F}_p \rightarrow \mathbb{F}_p$ are two degree-d polynomials that are unequal (and $d \ll p$), then with high probability, their inequality can be seen by querying them at a random point:

$$\Pr_{x \in \mathbb{F}_p} \left[q(x) = q'(x) \right] \leq \frac{d}{p}.$$

Let me now give a brief impression of how one proves Theorem 51, or at least the simpler result coNP $\subseteq$ IP. Let $\varphi(x_1, \ldots, x_n)$ be, say, a 3SAT formula that Merlin wants to convince Arthur is unsatisfiable. Then Arthur first lifts φ to a multivariate polynomial $q : \mathbb{F}_p^n \rightarrow \mathbb{F}_p$, of degree $d \leq |\varphi|$ (where $|\varphi|$ is the size of φ), over the finite field $\mathbb{F}_p$, for some $p \gg 2^n$. Merlin's task is equivalent to convincing Arthur of the following equation:

$$\sum_{x_1, \ldots, x_n \in \{0,1\}} q(x_1, \ldots, x_n) = 0.$$

To achieve this, Merlin first sends Arthur the coefficients of a univariate polynomial $q_1 : \mathbb{F}_p \rightarrow \mathbb{F}_p$. Merlin claims that q_1 satisfies

$$q_1(x_1) = \sum_{x_2, \ldots, x_n \in \{0,1\}} q(x_1, x_2, \ldots, x_n), \tag{1}$$

and also satisfies $q_1(0) + q_1(1) = 0$. Arthur can easily check the latter equation for himself. To check equation (1), Arthur picks a random value $r_1 \in \mathbb{F}_p$ for x_1 and sends it to Merlin. Then

[49] This assumes that we restrict attention to chess games that end after a "reasonable" number of moves, as would be normal in tournament play because of the time limits. Formally, if we consider a generalization of chess to $n \times n$ boards, then deciding the win from a given board position is in PSPACE (and in fact PSPACE-complete [233]), and hence in IP, provided we declare a game a draw if it's gone on for $p(n)$ moves, for some polynomial p. Without the restriction to $n^{O(1)}$ moves, chess is EXP-complete [92].

Merlin replies with a univariate polynomial q_2, for which he claims that

$$q_2(x_2) = \sum_{x_3,\ldots,x_n \in \{0,1\}} q(r_1, x_2, x_3, \ldots, x_n).$$

Arthur checks that $q_2(0) + q_2(1) = q_1(r_1)$, then picks a random value $r_2 \in \mathbb{F}_p$ for x_2 and sends it to Merlin, and so on. Finally, Arthur checks that q_n is indeed the univariate polynomial obtained by starting from the arithmetization of φ, then fixing $x_1, \ldots, x_{n-1}$ to $r_1, \ldots, r_{n-1}$ respectively. The Fundamental Theorem of Algebra ensures that, if Merlin lied at any point in the protocol, then with high probability at least one of Arthur's checks will fail.

Now, to return to the question that interests us: why does this protocol escape the relativization barrier? The short answer is: because if the Boolean formula φ involved oracle gates, then we wouldn't have been able to arithmetize φ. By arithmetizing φ, we did something "deeper" with it, more dependent on its structure, than simply evaluating φ on various Boolean inputs (which would have continued to work fine had an oracle been involved).

Arithmetization made sense because φ was built out of AND and OR and NOT gates, which we were able to reinterpret arithmetically. But how would we arithmetically reinterpret an oracle gate?

6.3.2 Hybrid Circuit Lower Bounds

To recap, $\mathsf{PSPACE} \subseteq \mathsf{IP}$ is a non-relativizing inclusion of complexity classes. But can we leverage that achievement to prove non-relativizing *separations* between complexity classes, with an eye toward $\mathsf{P} \neq \mathsf{NP}$? Certainly, by combining $\mathsf{IP} = \mathsf{PSPACE}$ with the Space Hierarchy Theorem (which implies $\mathsf{SPACE}\left(n^k\right) \neq \mathsf{PSPACE}$ for every fixed k), we get that $\mathsf{IP} \not\subset \mathsf{SPACE}\left(n^k\right)$ for every fixed k. Likewise, by combining $\mathsf{IP} = \mathsf{PSPACE}$ with Theorem 40 (that PSPACE does not have circuits of size n^k for fixed k), we get that IP doesn't have circuits of size n^k either. Furthermore, both of these separations can be shown to be non-relativizing, using techniques from [60]. But can we get more interesting separations?

The key to doing so turns out to be a beautiful corollary of the $\mathsf{IP} = \mathsf{PSPACE}$ theorem. To state the corollary, we need one more complexity class: MA (Merlin-Arthur) is a probabilistic generalization of NP. It's defined as the class of languages $L \subseteq \{0,1\}^*$ for which there exists a probabilistic polynomial-time verifier M, and a polynomial p, such that for all inputs $x \in \{0,1\}^*$:

- If $x \in L$ then there exists a witness string $w \in \{0,1\}^{p(|x|)}$ such that $M(x,w)$ accepts with probability at least $2/3$ over its internal randomness.
- If $x \notin L$, then $M(x,w)$ rejects with probability at least $2/3$ over its internal randomness, for all w.

Clearly MA contains NP and BPP. It can also be shown that $\mathsf{MA} \subseteq \Sigma_2^{\mathsf{P}} \cap \Pi_2^{\mathsf{P}}$ and that $\mathsf{MA} \subseteq \mathsf{PP}$, where PP is the counting class from Section 2.2.6. Now, here's the corollary of Theorem 51:

Corollary 52 *If* $\mathsf{PSPACE} \subset \mathsf{P/poly}$*, then* $\mathsf{PSPACE} = \mathsf{MA}$*.*

Proof. Suppose $\mathsf{PSPACE} \subset \mathsf{P/poly}$, let $L \in \mathsf{PSPACE}$, and let x be an input in L. Then as an MA witness proving that $x \in L$, Merlin simply sends Arthur a description of a polynomial-size circuit

C that simulates the PSPACE prover, in an interactive protocol that convinces Arthur that $x \in L$. (Here we use one additional fact about Theorem 51, beyond the mere fact that IP = PSPACE: that, in the protocol, Merlin can run a PSPACE algorithm to decide which message to send next.) Then Arthur simulates the protocol, using C to compute Merlin's responses to his random challenges, and accepts if and only if the protocol does. Hence $L \in \mathsf{MA}$. ■

Likewise:

Corollary 53 *If* $\mathsf{P}^{\#\mathsf{P}} \subset \mathsf{P/poly}$, *then* $\mathsf{P}^{\#\mathsf{P}} = \mathsf{MA}$.

(Again, here we use the observation that, in the protocol proving that $\mathsf{P}^{\#\mathsf{P}} \subseteq \mathsf{IP}$, Merlin can run a $\mathsf{P}^{\#\mathsf{P}}$ algorithm to decide which message to send next.)

Let's now see how we can use these corollaries of IP = PSPACE to prove new circuit lower bounds. Let $\mathsf{MA}_{\mathsf{EXP}}$ be "the exponential-time version of MA," with a $2^{p(n)}$-size witness that can be probabilistically verified in $2^{p(n)}$ time: in other words, the class that is to MA as NEXP is to NP. Then:

Theorem 54 (Buhrman-Fortnow-Thierauf [60]) $\mathsf{MA}_{\mathsf{EXP}} \not\subset \mathsf{P/poly}$.

Proof. Suppose by contradiction that $\mathsf{MA}_{\mathsf{EXP}} \subset \mathsf{P/poly}$. Then certainly $\mathsf{PSPACE} \subset \mathsf{P/poly}$, which means that PSPACE = MA by Corollary 52. By a padding argument (see Proposition 17), this means that $\mathsf{EXPSPACE} = \mathsf{MA}_{\mathsf{EXP}}$. But we already saw in Theorem 39 that $\mathsf{EXPSPACE} \not\subset \mathsf{P/poly}$, and therefore $\mathsf{MA}_{\mathsf{EXP}} \not\subset \mathsf{P/poly}$ as well. ■

Note in particular that if we could prove MA = NP, then we'd also have $\mathsf{MA}_{\mathsf{EXP}} = \mathsf{NEXP}$ by padding, and hence $\mathsf{NEXP} \not\subset \mathsf{P/poly}$ by Theorem 54. This provides another example of how derandomization can lead to circuit lower bounds, a theme mentioned in Section 6.

A second example involves the class PP.

Theorem 55 (Vinodchandran [252]) *For every fixed* k, *there is a language in* PP *that does not have circuits of size* n^k.

Proof. Fix k, and suppose by contradiction that PP has circuits of size n^k. Then in particular, $\mathsf{PP} \subset \mathsf{P/poly}$, so $\mathsf{P}^{\mathsf{PP}} = \mathsf{P}^{\#\mathsf{P}} \subset \mathsf{P/poly}$, so $\mathsf{P}^{\#\mathsf{P}} = \mathsf{PP} = \mathsf{MA}$ by Corollary 53. But we noted in Section 6.1 that Σ_2^{P} does not have circuits of size n^k. And $\Sigma_2^{\mathsf{P}} \subseteq \mathsf{P}^{\#\mathsf{P}}$ by Toda's Theorem (Theorem 13), so $\mathsf{P}^{\#\mathsf{P}}$ doesn't have circuits of size n^k either. Therefore neither does PP.[50] ■

As a final example, Santhanam [214] showed the following (we omit the proof).

Theorem 56 (Santhanam [214]) *For every fixed* k, *there is an* MA *"promise problem"*[51] *that does not have circuits of size* n^k.

[50] Actually, for this proof one does not really need either Toda's Theorem, *or* the slightly-nontrivial result that Σ_2^{P} does not have circuits of size n^k. Instead, one can just argue directly that at any rate, $\mathsf{P}^{\#\mathsf{P}}$ does not have circuits of size n^k, using a slightly more careful version of the argument of Theorem 39. For details see Aaronson [4].

[51] In complexity theory, a *promise problem* is a pair of subsets $\Pi_{\mathrm{YES}}, \Pi_{\mathrm{NO}} \subseteq \{0,1\}^*$ with $\Pi_{\mathrm{YES}} \cap \Pi_{\mathrm{NO}} = \varnothing$. An algorithm solves the problem if it accepts all inputs in Π_{YES} and rejects all inputs in Π_{NO}. Its behavior on inputs neither in Π_{YES} nor Π_{NO} (i.e., inputs that "violate the promise") can be arbitrary. A typical example of a promise problem is: given a Boolean circuit C, decide whether C accepts at least 2/3 of all inputs $x \in \{0,1\}^n$ or at most 1/3 of them, promised that one of those is true. This problem is in BPP (or technically, PromiseBPP). The role of the promise here is to get rid of those inputs for which random sampling would accept with probability between 1/3 and 2/3, violating the definition of BPP.

The above results clearly evade the natural proofs barrier, because they give lower bounds against strong circuit classes such as $\mathsf{P/poly}$, or the set of all size-n^k circuits for fixed k. This is not so surprising when we observe that the proofs build on the simpler results from Section 6.1, which already used diagonalization to evade the natural proofs barrier.

What's more interesting is that these results *also* evade the relativization barrier. Of course, one might guess as much, after noticing that the proofs use the non-relativizing $\mathsf{IP} = \mathsf{PSPACE}$ theorem. But to show rigorously that the circuit lower bounds *themselves* fail to relativize, one needs to construct oracles relative to which the circuit lower bounds are false. This is done by the following results, whose somewhat elaborate proofs we omit:

Theorem 57 (Buhrman-Fortnow-Thierauf [60]) *There exists an oracle A such that $\mathsf{MA}_{\mathsf{EXP}}^A \subset \mathsf{P}^A/\mathsf{poly}$.*

Theorem 58 (Aaronson [4]) *There exists an oracle A relative to which all languages in PP have linear-sized circuits.*

The proofs of both of these results also easily imply that there exists an oracle relative to which all MA promise problems have linear-sized circuits.

The bottom line is that, by combining non-relativizing results like $\mathsf{IP} = \mathsf{PSPACE}$ with non-naturalizing results like $\mathsf{EXPSPACE} \not\subset \mathsf{P/poly}$, we can prove interesting circuit lower bounds that neither relativize *nor* naturalize. So then why couldn't we keep going, and use similar techniques to prove $\mathsf{NEXP} \not\subset \mathsf{P/poly}$, or even $\mathsf{P} \neq \mathsf{NP}$? Is there a third barrier, to which even the arithmetization-based lower bounds are subject?

6.3.3 The Algebrization Barrier

In 2008, Avi Wigderson and I [10] showed that, alas, there's a third barrier. In particular, while the arithmetic techniques used to prove $\mathsf{IP} = \mathsf{PSPACE}$ do evade relativization, they crash up against a modified version of relativization that's "wise" to those techniques. We called this modified barrier the *algebraic relativization* or *algebrization* barrier. We then showed that, in order to prove $\mathsf{P} \neq \mathsf{NP}$—or for that matter, even to prove $\mathsf{NEXP} \not\subset \mathsf{P/poly}$, or otherwise go even slightly beyond the results of Section 6.3.2—we'd need techniques that evade the algebrization barrier (and *also*, of course, evade natural proofs).

In more detail, we can think of an oracle as just an infinite collection of Boolean functions, $f_n : \{0,1\}^n \rightarrow \{0,1\}$ for each n. Now, by an *algebraic oracle*, we mean an oracle that provides access not only to f_n for each n, but also to a low-degree extension $\widetilde{f_n} : \mathbb{F}^n \rightarrow \mathbb{F}$ of f_n over some large finite field $\mathbb{F}$. This extension must have the property that $\widetilde{f_n}(x) = f_n(x)$ for all $x \in \{0,1\}^n$, and it must be a polynomial of low degree—say, at most $2n$. But such extensions always exist,[52] and querying them *outside* the Boolean cube $\{0,1\}^n$ might help even for learning about the Boolean part f_n.

The point of algebraic oracles is that they capture what we could do if we had a formula or circuit for f_n, and were willing to evaluate it not only on Boolean inputs, but on non-Boolean ones as well, in the manner of $\mathsf{IP} = \mathsf{PSPACE}$. In particular, we saw in Section 6.3.1 that, given (say) a 3SAT formula φ, we can "lift" φ to a low-degree polynomial $\widetilde{\varphi}$ over a finite field $\mathbb{F}$ by reinterpreting

[52] Indeed, every f_n has an extension to a degree-n polynomial, namely a *multilinear* one (in which no variable is raised to a higher power than 1): for example, $\mathrm{OR}(x,y) = x + y - xy$.

the AND, OR, and NOT gates in terms of field addition and multiplication. So if we're trying to capture the power of arithmetization relative to an oracle function f_n, then it stands to reason that we should also be allowed to lift f_n.

Once we do so, we find that the non-relativizing results based on arithmetization, such as $\mathsf{IP} = \mathsf{PSPACE}$, *relativize with respect to algebraic oracles* (or "algebrize"). That is:

Theorem 59 $\mathsf{IP}^{\widetilde{A}} = \mathsf{PSPACE}^{\widetilde{A}}$ *for all algebraic oracles* $\widetilde{A}$. *Likewise,* $\mathsf{PSPACE}^{\widetilde{A}} \subset \mathsf{P}^{\widetilde{A}}/\mathsf{poly}$ *implies* $\mathsf{PSPACE}^{\widetilde{A}} = \mathsf{MA}^{\widetilde{A}}$ *for all algebraic oracles* $\widetilde{A}$, *and so on for all the interactive proof results.*

The intuitive reason is that, any time (say) Arthur needs to arithmetize a formula φ containing A-oracle gates in an interactive protocol, he can handle non-Boolean inputs to the A-oracle gates by calling $\widetilde{A}$.

As a consequence of Theorem 59, the circuit lower bounds of Section 6.3.2 are algebrizing as well: for example, for all algebraic oracles $\widetilde{A}$, we have $\mathsf{MA}^{\widetilde{A}}_{\mathsf{EXP}} \not\subset \mathsf{P}^{\widetilde{A}}/\mathsf{poly}$, and $\mathsf{PP}^{\widetilde{A}}$ does not have size-n^k circuits with $\widetilde{A}$-oracle gates.

Admittedly, the original paper of Aaronson and Wigderson [10] only managed to prove a weaker version of Theorem 59. It showed, for example, that for all algebraic oracles $\widetilde{A}$, we have $\mathsf{PSPACE}^{A} \subseteq \mathsf{IP}^{\widetilde{A}}$, and $\mathsf{MA}^{\widetilde{A}}_{\mathsf{EXP}} \not\subset \mathsf{P}^{A}/\mathsf{poly}$. As a result, it had to define algebrization in a convoluted way, where some complexity classes received the algebraic oracle $\widetilde{A}$ while others received only the "original" oracle A, and which class received which depended on what kind of result one was talking about (e.g., an inclusion or a separation). Shortly afterward, Impagliazzo, Kabanets, and Kolokolova [123] fixed this defect of algebrization, proving Theorem 59 even when all classes receive the same algebraic oracle $\widetilde{A}$, but only at the cost of jettisoning Aaronson and Wigderson's conclusion that any proof of $\mathsf{NEXP} \not\subset \mathsf{P}/\mathsf{poly}$ will require non-algebrizing techniques. Very recently, Aydınlıoğlu and Bach [192] showed how to get the best of both worlds, with a uniform definition of algebrization *and* the conclusion about NEXP vs. P/poly.

In any case, the main point of [10] was that to prove $\mathsf{P} \neq \mathsf{NP}$, or otherwise go further than the circuit lower bounds of Section 6.3.2, we'll need non-algebrizing techniques: techniques that fail to relativize in a "deeper" way than $\mathsf{IP} = \mathsf{PSPACE}$ fails to relativize. Let's see why this is true for $\mathsf{P} \neq \mathsf{NP}$.

Theorem 60 (Aaronson-Wigderson [10]) *There exists an algebraic oracle* $\widetilde{A}$ *such that* $\mathsf{P}^{\widetilde{A}} = \mathsf{NP}^{\widetilde{A}}$. *As a consequence, any proof of* $\mathsf{P} \neq \mathsf{NP}$ *will require non-algebrizing techniques.*

Proof. We can just let A be any PSPACE-complete language, and then let $\widetilde{A}$ be its unique extension to a collection of multilinear polynomials over $\mathbb{F}$ (that is, polynomials in which no variable is ever raised to a higher power than 1). The key observation is that the multilinear extensions are themselves computable in PSPACE. So we get a PSPACE-complete oracle $\widetilde{A}$, which collapses P and NP for the same reason as in the original argument of Baker, Gill, and Solovay [36] (see Theorem 42). ■

Likewise, Aaronson and Wigderson [10] showed that any proof of $\mathsf{P} = \mathsf{NP}$, or even $\mathsf{P} = \mathsf{BPP}$, would need non-algebrizing techniques. They also proved the following somewhat harder result, whose proof we omit.

Theorem 61 ([10]) *There exists an algebraic oracle* $\widetilde{A}$ *such that* $\mathsf{NEXP}^{\widetilde{A}} \subset \mathsf{P}^{\widetilde{A}}/\mathsf{poly}$. *As a consequence, any proof of* $\mathsf{NEXP} \not\subset \mathsf{P}/\mathsf{poly}$ *will require non-algebrizing techniques.*

Note that this explains almost exactly why progress stopped where it did: $\mathsf{MA_{EXP}} \not\subset \mathsf{P/poly}$ can be proved with algebrizing techniques, but $\mathsf{NEXP} \not\subset \mathsf{P/poly}$ can't be.

I should mention that Impagliazzo, Kabanets, and Kolokolova [123] gave a logical interpretation of algebrization, extending the logical interpretation of relativization given by Arora, Impagliazzo, and Vazirani [29]. Impagliazzo et al. show that the algebrizing statements can be seen as all those statements that follow from "algebrizing axioms for computation," which include basic closure properties, *and also* the ability to lift any Boolean computation to a larger finite field. Statements like $\mathsf{P} \neq \mathsf{NP}$ are then provably independent of the algebrizing axioms.

6.4 Ironic Complexity Theory

There's one technique that's had some striking recent successes in proving circuit lower bounds, and that bypasses the natural proofs, relativization, *and* algebrization barriers. This technique might be called "ironic complexity theory." It uses the existence of surprising algorithms in one setting to show the *non*existence of algorithms in another setting. It thus reveals a "duality" between upper and lower bounds, and reduces the problem of proving impossibility theorems to the much better-understood task of designing efficient algorithms.[53]

At a conceptual level, it's not hard to see how algorithms can lead to lower bounds. For example, suppose someone discovered a way to verify arbitrary exponential-time computations efficiently, thereby proving $\mathsf{NP} = \mathsf{EXP}$. Then as an immediate consequence of the Time Hierarchy Theorem ($\mathsf{P} \neq \mathsf{EXP}$), we'd get $\mathsf{P} \neq \mathsf{NP}$. Or suppose someone discovered that every language in P had linear-size circuits. Then $\mathsf{P} = \mathsf{NP}$ would imply that every language in PH had linear-size circuits—but since we know that's not the case (see Section 6.1), we could again conclude that $\mathsf{P} \neq \mathsf{NP}$. Conversely, if someone proved $\mathsf{P} = \mathsf{NP}$, that wouldn't be a total disaster for lower bounds research: at least it would immediately imply $\mathsf{EXP} \not\subset \mathsf{P/poly}$ (via $\mathsf{EXP} = \mathsf{EXP}^{\mathsf{NP}^{\mathsf{NP}}}$), and the existence of languages in P and NP that don't have linear-size circuits!

Examples like this can be multiplied, but there's an obvious problem with them: they each show a separation, but only assuming a collapse that's considered extremely unlikely to happen. However, recently researchers have managed to use surprising algorithms that *do* exist, and collapses that *do* happen, to achieve new lower bounds. In this section I'll give two examples.

6.4.1 Time-Space Tradeoffs

At the moment, no one can prove that solving 3SAT requires more than linear time (let alone exponential time!), on realistic models of computation like random-access machines.[54] Nor can anyone prove that solving 3SAT requires more than $O(\log n)$ bits of memory. But the situation isn't completely hopeless: at least we can prove there's no algorithm for 3SAT that uses both linear time *and* logarithmic memory! Indeed, we can do better than that.

A bit of background: just as one can scale PSPACE up to $\mathsf{EXPSPACE}$ and so on, one can also scale PSPACE down to $\mathsf{LOGSPACE}$, which is the class of languages L decidable by a Turing machine that uses only $O(\log n)$ bits of read/write memory, in addition to a read-only memory that stores

[53] Indeed, the hybrid circuit lower bounds of Section 6.3.2 could already be considered examples of ironic complexity theory. In this section, we discuss other examples.

[54] On unrealistic models such as one-tape Turing machines, one can prove up to $\Omega(n^2)$ lower bounds for 3SAT and many other problems (even recognizing palindromes), but only by exploiting the fact that the tape head needs to waste a lot of time moving back and forth across the input.

the n-bit input itself. We have $\mathsf{LOGSPACE} \subseteq \mathsf{P}$, for the same simple reason why $\mathsf{PSPACE} \subseteq \mathsf{EXP}$ (see Proposition 15). We also have $\mathsf{LOGSPACE} \neq \mathsf{PSPACE}$ by the Space Hierarchy Theorem. On the other hand, no one has proven even that $\mathsf{LOGSPACE} \neq \mathsf{NP}$.

Now, a "time-space tradeoff theorem" shows that any algorithm to solve some problem must use *either* more than T time or else more than S space. The first such theorem for 3SAT was proved by Fortnow [86], who showed that no random-access machine can solve 3SAT simultaneously in $n^{1+o(1)}$ time and $n^{1-\varepsilon}$ space, for any $\varepsilon > 0$. Later, Lipton and Viglas [165] gave a different tradeoff involving a striking exponent; I'll use their result as my running example in this section:

Theorem 62 (Lipton-Viglas [165]) *No random-access machine can solve* 3SAT *simultaneously in $n^{\sqrt{2}-\varepsilon}$ time and $n^{o(1)}$ space, for any $\varepsilon > 0$.*

Here, a "random-access machine" means a machine that can access an arbitrary memory location in $O(1)$ time, as usual in practical programming. This makes Theorem 62 *stronger* than one might have assumed: it holds not merely for unrealistically weak models such as Turing machines, but for "realistic" models as well.[55] Also, again, "$n^{o(1)}$ space" means that we get the n-bit 3SAT instance itself in a read-only memory, and also get $n^{o(1)}$ bits of auxiliary read/write memory.

While Theorem 62 is obviously a far cry from $\mathsf{P} \neq \mathsf{NP}$, it does rely essentially on 3SAT being NP-complete: we don't yet know how to prove analogous results for matching, linear programming, or other natural problems in P.[56] This makes Theorem 62 fundamentally different from (say) the PARITY $\notin \mathsf{AC}^0$ result of Section 6.2.3.

Let $\mathsf{DTISP}(T, S)$ be the class of languages decidable by an algorithm, running on a RAM machine, that uses $O(T)$ time and $O(S)$ space. Then Theorem 62 can be stated more succinctly as

$$3\text{SAT} \notin \mathsf{DTISP}\left(n^{\sqrt{2}-\varepsilon}, n^{o(1)}\right)$$

for all $\varepsilon > 0$.

At a high level, Theorem 62 is proved by assuming the opposite, and then deriving stranger and stranger consequences until we ultimately get a contradiction with the Nondeterministic Time Hierarchy Theorem (Theorem 37). There are three main ideas that go into this. The first idea is a tight version of the Cook-Levin Theorem (Theorem 2). In particular, one can show, not merely that 3SAT is NP-complete, but that 3SAT is complete for $\mathsf{NTIME}(n)$ (that is, nondeterministic linear-time on a RAM machine) under nearly linear-time reductions—and moreover, that each individual bit of the 3SAT instance is computable quickly, say in $\log^{O(1)} n$ time. This means that, to prove Theorem 62, it suffices to prove a non-containment of complexity classes:

$$\mathsf{NTIME}(n) \not\subset \mathsf{DTISP}\left(n^{\sqrt{2}-\varepsilon}, n^{o(1)}\right)$$

[55] Some might argue that Turing machines are *more* realistic than RAM machines, since Turing machines take into account that signals can propagate only at a finite speed, whereas RAM machines don't! However, RAM machines are closer to what's assumed in practical algorithm development, whenever memory latency is small enough to treated as a constant.

[56] On the other hand, by proving size-depth tradeoffs for so-called *branching programs*, researchers have been able to obtain time-space tradeoffs for certain special problems in P. Unlike the 3SAT tradeoffs, the branching program tradeoffs involve only *slightly* superlinear time bounds; on the other hand, they really do represent a fundamentally different way to prove time-space tradeoffs, one that makes no appeal to NP-completeness, diagonalization, or hierarchy theorems. As one example, in 2000 Beame et al. [39], building on earlier work by Ajtai [18], used branching programs to prove the following: there exists a problem in P, based on binary quadratic forms, for which any RAM algorithm (even a nonuniform one) that uses $n^{1-\Omega(1)}$ space must also use $\Omega\left(n \cdot \sqrt{\log n / \log\log n}\right)$ time.

for all $\varepsilon > 0$.

The second idea is called "trading time for alternations." Consider a deterministic computation that runs for T steps and uses S bits of memory. Then we can "chop the computation up" into k blocks, $B_1, \ldots, B_k$, of T/k steps each. The statement that the computation accepts is then equivalent to the statement that *there exist* S-bit strings $x_0, \ldots, x_k$, such that

(i) x_0 is the computation's initial state,

(ii) *for all* $i \in \{1, \ldots, k\}$, the result of starting in state x_{i-1} and then running for T/k steps is x_i, and

(iii) x_k is an accepting state.

We can summarize this as

$$\mathsf{DTISP}\left(T, S\right) \subseteq \Sigma_2\mathsf{TIME}\left(Sk + \frac{T}{k}\right),$$

where the Σ_2 means that we have two alternating quantifiers: an existential quantifier over $x_1, \ldots, x_k$, followed by a universal quantifier over i. Choosing $k := \sqrt{T/S}$ to optimize the bound then gives us

$$\mathsf{DTISP}\left(T, S\right) \subseteq \Sigma_2\mathsf{TIME}\left(\sqrt{TS}\right).$$

So in particular,

$$\mathsf{DTISP}\left(n^c, n^{o(1)}\right) \subseteq \Sigma_2\mathsf{TIME}\left(n^{c/2+o(1)}\right).$$

The third idea is called "trading alternations for time." If we assume by way of contradiction that

$$\mathsf{NTIME}\left(n\right) \subseteq \mathsf{DTISP}\left(n^c, n^{o(1)}\right) \subseteq \mathsf{TIME}\left(n^c\right),$$

then in particular, for all $b \geq 1$, we can add an existential quantifier to get

$$\Sigma_2\mathsf{TIME}\left(n^b\right) \subseteq \mathsf{NTIME}\left(n^{bc}\right).$$

So putting everything together, if we consider a constant $c > 1$, and use padding (as in Proposition 17) to talk about $\mathsf{NTIME}\left(n^2\right)$ rather than $\mathsf{NTIME}\left(n\right)$, then the starting assumption that 3SAT is solvable in $n^{c-\varepsilon}$ time and $n^{o(1)}$ space implies that

$$\begin{aligned}
\mathsf{NTIME}\left(n^2\right) &\subseteq \mathsf{DTISP}\left(n^{2c}, n^{o(1)}\right) \\
&\subseteq \Sigma_2\mathsf{TIME}\left(n^{c+o(1)}\right) \\
&\subseteq \mathsf{NTIME}\left(n^{c^2+o(1)}\right).
\end{aligned}$$

But if $c^2 < 2$, then this contradicts the Nondeterministic Time Hierarchy Theorem (Theorem 37). This completes the proof of Theorem 62. Notice that the starting hypothesis about 3SAT was applied not once but twice, which was how the final running time became n^{c^2}.

A proof of this general form, making a sequence of trades between running time and nondeterminism, is called an *alternating-trading proof*. Later, using a more involved alternating-trading proof, Fortnow and van Melkebeek [89] improved Theorem 62, to show that 3SAT can't be solved by a RAM machine using $n^{\phi-\varepsilon}$ time and $n^{o(1)}$ space, where $\phi = \frac{1+\sqrt{5}}{2} \approx 1.618$ is the golden ratio. Subsequently Williams [256] improved the time bound still further to $n^{\sqrt{3}-\varepsilon}$, and then [258] to $n^{2\cos\pi/7-\varepsilon}$. In 2012, however, Buss and Williams [67] showed that no alternation-trading proof can possibly improve that exponent beyond the peculiar constant $2\cos\pi/7 \approx 1.801$. There have been many related time-space tradeoff results, including for #P-complete and PSPACE-complete problems, but I won't cover them here (see van Melkebeek [169] for a survey).

Alternation-trading has had applications in complexity theory other than to time-space tradeoffs. In particular, it played a key role in a celebrated 1983 result of Paul, Pippenger, Szemeredi, and Trotter [195], whose statement is tantalizingly similar to $\mathsf{P} \neq \mathsf{NP}$.

Theorem 63 (Paul et al. [195]) $\mathsf{TIME}(n) \neq \mathsf{NTIME}(n)$*, if we define these classes using multiple-tape Turing machines.*

In this case, the key step was to show, via a clever combinatorial argument involving "pebble games," that for multi-tape Turing machines, deterministic linear time can be simulated in $\Sigma_4\mathsf{TIME}(f(n))$, for some f that's *slightly* sublinear. This, combined with the assumption $\mathsf{TIME}(n) = \mathsf{NTIME}(n)$, is then enough to produce a contradiction with a time hierarchy theorem.

What can we say about barriers? All the results mentioned above clearly evade the natural proofs barrier, because they ultimately rely on diagonalization, and (more to the point) because classes like $\mathsf{TIME}(n)$ and $\mathsf{DTISP}\left(n^{\sqrt{2}}, n^{o(1)}\right)$ contain plausible pseudorandom function candidates. Whether they evade the relativization barrier (let alone algebrization) is a trickier question; it depends on subtle details of the oracle access mechanism. There are some definitions of the classes $\mathsf{TIME}(n)^A$, $\mathsf{DTISP}(T,S)^A$, and so on under which these results relativize, and others under which they don't: for details, see for example Moran [173].

On the definitions that cause these results *not* to relativize, the explanation for how is that the proofs "look inside" the operations of a RAM machine or a multi-tape Turing machine *just enough* for something to break down if certain kinds of oracle calls are present. To illustrate, in the proof of Theorem 62 above, we nondeterministically guessed the complete state of the machine at various steps in its execution, taking advantage of the fact that the state was an $n^{o(1)}$-bit string. This wouldn't have worked had there been an n-bit query written onto an oracle tape (even if the oracle tape were write-only). Likewise, in the proof of Theorem 63, the combinatorial pebble arguments use specific properties of multi-tape Turing machines that might fail for RAM machines, let alone for oracle machines.

Because their reasons for failing to relativize have nothing to do with lifting to large finite fields, I conjecture that, with some oracle access mechanisms, Theorems 62 and 63 would also be non-algebrizing. But this remains to be shown.

6.4.2 NEXP $\not\subset$ ACC

In Section 6.2.4, we saw how Smolensky [229] and Razborov [208] proved strong lower bounds against the class $\mathsf{AC}^0[p]$, or constant-depth, polynomial-size circuits of AND, OR, NOT, and MOD p gates, where p is prime. This left the frontier of circuit lower bounds as $\mathsf{AC}^0[m]$, where m is composite.

Meanwhile, we saw in Section 6.3 how Buhrman, Fortnow, and Thierauf [60] proved that $\mathsf{MA_{EXP}} \not\subset \mathsf{P/poly}$, but how this can't be extended even to $\mathsf{NEXP} \not\subset \mathsf{P/poly}$ using algebrizing techniques. Indeed, it remains open even to prove $\mathsf{NEXP} \not\subset \mathsf{TC}^0$.

This state of affairs—and its continuation for decades—helps to explain why many theoretical computer scientists were electrified when Ryan Williams proved the following in 2011.

Theorem 64 (Williams [265]) $\mathsf{NEXP} \not\subset \mathsf{ACC}$ *(and indeed* $\mathsf{NTIME}(2^n) \not\subset \mathsf{ACC}$*), where* ACC *is the union of* $\mathsf{AC}^0[m]$ *over all constants* m*.*[57]

If we compare it against the ultimate goal of proving $\mathsf{NP} \not\subset \mathsf{P/poly}$, Theorem 64 looks laughably weak: it shows only that Nondeterministic Exponential Time, a class vastly larger than NP, is not in ACC, a circuit class vastly smaller than $\mathsf{P/poly}$. But a better comparison is against where we were before. The proof of Theorem 64 was noteworthy not only because it defeats all the known barriers (relativization, algebrization, and natural proofs), but also because it brings together almost *all* known techniques in Boolean circuit lower bounds, including diagonalization, the polynomial method, interactive proof results, and ironic complexity theory. So it's worth at least sketching the elaborate proof, so we can see how a lower bound at the current frontier operates. (For further details, I recommend two excellent expository articles by Williams himself [259, 263].)

At a stratospherically high level, the proof of Theorem 64 is built around the Nondeterministic Time Hierarchy Theorem, following a program that Williams had previously laid out in [261]. More concretely, we assume that $\mathsf{NTIME}(2^n) \subset \mathsf{ACC}$. We then use that assumption to show that $\mathsf{NTIME}(2^n) = \mathsf{NTIME}\left(2^n/n^k\right)$ for some positive k: a slight speedup of nondeterministic machines, but enough to achieve a contradiction with Theorem 37.

How do we use the assumption $\mathsf{NTIME}(2^n) \subset \mathsf{ACC}$ to violate the Nondeterministic Time Hierarchy Theorem? The key to this—and this is where "ironic complexity theory" enters the story—is a faster-than-brute-force algorithm for a problem called ACCSAT. Here we're given as input a description of ACC circuit C, and want to decide whether there exists an input $x \in \{0,1\}^n$ such that $C(x) = 1$. The core of Williams's proof is the following straightforwardly algorithmic result.

Lemma 65 (Williams [265]) *There's a deterministic algorithm that solves* ACCSAT*, for* ACC *circuits of depth* d *with* n *inputs, in* $2^{n-\Omega(n^\delta)}$ *time, for some constant* $\delta > 0$ *that depends on* d*.*

The proof of Lemma 65 is itself a combination of several ideas. First, one appeals to a powerful structural result of Yao [270], Beigel-Tarui [40], and Allender-Gore [24] from the 1990s, which shows that functions in ACC are representable in terms of low-degree polynomials.

Lemma 66 ([270, 40, 24]) *Let* $f : \{0,1\}^n \to \{0,1\}$ *be computable by an* $\mathsf{AC}^0[m]$ *circuit of size* s *and depth* d*. Then* $f(x)$ *can be expressed as* $g(p(x))$*, where* $p : \{0,1\}^n \to \mathbb{N}$ *is a polynomial of degree* $\log^{O(1)} s$ *that's a sum of* $\exp\left(\log^{O(1)} s\right)$ *monomials with coefficients of* 1*, and* $g : \mathbb{N} \to \{0,1\}$ *is some efficiently computable function. Moreover, this conversion can be done in* $\exp\left(\log^{O(1)} s\right)$ *time. (Here the constant in the big-O depends on both* d *and* m*.)*

[57] We can also allow MOD-m_1 gates, MOD-m_2 gates, etc. in the same circuit; this is equivalent to $\mathsf{AC}^0[\gcd(m_1, m_2, \ldots)]$. On the other hand, if we allow MOD-m gates for *non*-constant m (and in particular, for m growing polynomially with n), then we jump up to TC^0.

The proof of Lemma 66 uses some elementary number theory, and is closely related to the polynomial method from Section 6.2.4, by which one shows that any $\mathsf{AC}^0[p]$ function can be approximated by a low-degree polynomial over the finite field $\mathbb{F}_p$.[58]

Next, one devises a faster-than-brute-force algorithm that, given a function $g(p(x))$ as above, decides whether there exists an $x \in \{0,1\}^n$ such that $g(p(x)) = 1$. The first step is to give an algorithm that constructs a table of all 2^n values of $p(x)$, for all the 2^n possible values of x, in $\left(2^n + s^{O(1)}\right) n^{O(1)}$ time, rather than the $O(2^n s)$ time that one would need naïvely. (In other words, this algorithm uses only $n^{O(1)}$ time on average per entry in the table, rather than $O(s)$ time—an improvement if s is superpolynomial.) Here there are several ways to go: one can use a fast rectangular matrix multiplication algorithm due to Coppersmith [75], but one can also just use a dynamic programming algorithm reminiscent of the Fast Fourier Transform.

Now, by combining this table-constructing algorithm with Lemma 66, we can immediately solve ACCSAT, for an ACC circuit of size $s = 2^{n^\delta}$, in $2^n n^{O(1)}$ time, which is better than the $O(2^n s)$ time that we would need naïvely. But this still isn't good enough to prove Lemma 65, which demands a $2^{n-\Omega(n^\delta)}$ algorithm. So there's a further trick: given an ACC circuit C of size $n^{O(1)}$, we first "shave off" n^δ of the n variables, building a new ACC circuit C' that takes as input the $n - n^\delta$ remaining variables, and that computes the OR of C over all 2^{n^δ} possible assignments to the n^δ shaved variables.[59] The new circuit C' has size $2^{O(n^\delta)}$, so we can construct the table, and thereby solve ACCSAT for C' (and hence for C), in time $2^{n-\Omega(n^\delta)}$.

Given Lemma 65, as well as the starting assumption $\mathsf{NTIME}(2^n) \subset \mathsf{ACC}$, there's still a lot of work to do to prove that $\mathsf{NTIME}(2^n) = \mathsf{NTIME}\left(2^n/n^k\right)$. Let me summarize the four main steps:

(1) We first use a careful, quantitative version of the Cook-Levin Theorem (Theorem 2), to reduce the problem of simulating an $\mathsf{NTIME}(2^n)$ machine to a problem called SUCCINCT3SAT. In that problem, we're given a circuit C whose truth table encodes an exponentially large 3SAT instance φ, and the problem is to decide whether or not φ is satisfiable. Indeed, we prove something stronger: the circuit C can be taken to be an AC^0 circuit.[60]

(2) We next appeal to a result of Impagliazzo, Kabanets, and Wigderson [124], which says that if $\mathsf{NEXP} \subset \mathsf{P/poly}$, then the satisfying assignments for satisfiable SUCCINCT3SAT instances can themselves be constructed by polynomial-size circuits.

(3) We massage the result (2) to get a conclusion about ACC: roughly speaking, if $\mathsf{NEXP} \subset \mathsf{ACC}$, then a satisfying assignment for an AC^0 SUCCINCT3SAT instance Φ can itself be constructed by an ACC circuit W. Furthermore, the problem of verifying that W does indeed encode a satisfying assignment for Φ can be solved in slightly less than 2^n time nondeterministically, if we use the fact (Lemma 65) that ACCSAT is solvable in $2^{n-\Omega(n^\delta)}$ time.

(4) Putting everything together, we get that $\mathsf{NTIME}(2^n)$ machines can be reduced to AC^0 SUCCINCT3SAT instances, which can then (assuming $\mathsf{NEXP} \subset \mathsf{ACC}$, and using the ACCSAT al-

[58] Interestingly, both the polynomial method and the proof of Lemma 66 are also closely related to the proof of Toda's Theorem (Theorem 13), that $\mathsf{PH} \subseteq \mathsf{P}^{\#\mathsf{P}}$.

[59] Curiously, this step can only be applied to the ACC circuits themselves, which of course allow OR gates. It can't be applied to the Boolean functions of low-degree polynomials that one derives from the ACC circuits.

[60] In Williams's original paper [265], this step required invoking the $\mathsf{NEXP} \subset \mathsf{ACC}$ assumption (and only yielded an ACC circuit). But subsequent improvements have made this step unconditional: see for example, Jahanjou, Miles, and Viola [127].

gorithm) be decided in $\mathsf{NTIME}\left(2^n/n^k\right)$ for some positive k. But that contradicts the Nondeterministic Time Hierarchy Theorem (Theorem 37).

Let me mention some improvements and variants of Theorem 64. Already in his original paper [265], Williams noted that the proof actually yields a stronger result, that $\mathsf{NTIME}\left(2^n\right)$ has no ACC circuits of "third-exponential" size: that is, size $f\left(n\right)$ where $f\left(f\left(f\left(n\right)\right)\right)$ grows exponentially. He also gave a second result, that $\mathsf{TIME}\left(2^n\right)^{\mathsf{NP}}$—that is, deterministic exponential time with an NP oracle—has no ACC circuits of size $2^{n^{o(1)}}$. More recently, Williams has extended Theorem 64 to show that $\mathsf{NTIME}\left(2^n\right)/1 \cap \mathsf{coNTIME}\left(2^n\right)/1$ (where the /1 denotes 1 bit of nonuniform advice) doesn't have ACC circuits of size $n^{\log n}$ [262], and also to show that even ACC circuits of size $n^{\log n}$ with threshold gates at the bottom layer can't compute all languages in NEXP [264].

At this point, I should step back and make some general remarks about the proof of Theorem 64 and the prospects for pushing it further. First of all, why did this proof only yield lower bounds for functions in the huge complexity class NEXP, rather than EXP or NP or even P? The short answer is that, in order to prove that a class $\mathcal{C}$ is not in ACC via this approach, we need to use the assumption $\mathcal{C} \subset \mathsf{ACC}$ to violate a hierarchy theorem for $\mathcal{C}$-like classes. But there's a bootstrapping problem: the mere fact that $\mathcal{C}$ *has* small ACC circuits doesn't imply that we can *find* those circuits in a $\mathcal{C}$-like class, in order to obtain the desired contradiction. When $\mathcal{C} = \mathsf{NEXP}$, we can use the nondeterministic guessing power of the NTIME classes simply to *guess* the small ACC circuits for NEXP, but even when $\mathcal{C} = \mathsf{EXP}$ this approach seems to break down.

A second question is: what in Williams's proof was specific to ACC? Here the answer is that the proof used special properties of ACC in one place only: namely, in the improved algorithm for ACCSat (Lemma 65). This immediately suggests a possible program to prove $\mathsf{NEXP} \not\subset \mathcal{C}$ for larger and larger circuit classes $\mathcal{C}$. For example, let TC^0Sat be the problem where we're given as input a TC^0 circuit C (that is, a neural network, or constant-depth circuit of threshold gates), and we want to decide whether there exists an $x \in \{0,1\}^n$ such that $C\left(x\right) = 1$. Then if we could solve TC^0Sat even slightly faster than brute force—say, in $O\left(2^n/n^k\right)$ time for some positive k—Williams's results would immediately imply $\mathsf{NEXP} \not\subset \mathsf{TC}^0$.[61] Likewise, recall from Section 2.1 that CircuitSat is the satisfiability problem for *arbitrary* Boolean circuits. If we had an $O\left(2^n/n^k\right)$ algorithm for CircuitSat, then Williams's results would imply the long-sought $\mathsf{NEXP} \not\subset \mathsf{P/poly}$.

Admittedly, one might be skeptical that faster-than-brute-force algorithms should even *exist* for problems like TC^0Sat and CircuitSat. But Williams and others have addressed that particular worry, by showing that circuit lower bounds for NEXP would follow even from faster-than-brute-force *derandomization* algorithms: that is, deterministic algorithms to find satisfying assignments under the assumption that a constant fraction of all assignments are satisfying. Obviously there's a fast randomized algorithm R under that assumption: namely, keep picking random assignments until you find one that works! Thus, to prove a circuit lower bound, "all we'd need to do" is give a nontrivial deterministic simulation of R—something that would necessarily exist under standard derandomization hypotheses (see Section 5.4.1). More precisely:

[61] Very recently, Kane and Williams [131] managed to give an explicit Boolean function that requires depth-2 threshold circuits with $\Omega\left(n^{3/2}/\log^3 n\right)$ gates. However, their argument doesn't proceed via a better-than-brute-force algorithm for depth-2 TC^0Sat. Even more recently, Chen, Santhanam, and Srinivasan [69] gave the first nontrivial algorithm for TC^0Sat with a slightly-superlinear number of wires. This wasn't enough for a new lower bound, but Chen et al. also used related ideas to show that Parity can't be computed even by polynomial-size AC^0 circuits with $n^{o(1)}$ threshold gates.

Theorem 67 (Williams [261], Santhanam-Williams [215]) *Suppose there's a deterministic algorithm, running in $2^n/f(n)$ time for any superpolynomial function f, to decide whether a polynomial-size Boolean circuit $C : \{0,1\}^n \rightarrow \{0,1\}$ has no satisfying assignments or at least 2^{n-2} satisfying assignments, promised that one of those is the case. Then $\mathsf{NEXP} \not\subset \mathsf{P/poly}$.*[62] *Likewise, if such an algorithm exists for TC^0 circuits, then $\mathsf{NEXP} \not\subset \mathsf{TC}^0$.*

Theorem 67 means that, besides viewing Williams's program as "ironic complexity theory," we can also view it as an instance of *circuit lower bounds through derandomization*, an idea discussed in Section 6.

In addition to derandomization, one could use faster algorithms for certain classic computational problems to prove circuit lower bounds. These algorithms might or might not exist, but it's at least plausible that they do. Thus, recall Theorem 23, which said that if CircuitSAT for circuits of depth $o(n)$ requires $(2-o(1))^n$ time, then EditDistance requires nearly quadratic time. In Section 5.1, we interpreted this to mean that there's a plausible hardness conjecture—albeit, one much stronger than $\mathsf{P} \neq \mathsf{NP}$—implying that the classic $O(n^2)$ algorithm for EditDistance is nearly optimal. But there's a different way to interpret the same connection: namely, if EditDistance *were* solvable in less than $\sim n^2$ time, then various satisfiability problems would be solvable in less than $\sim 2^n$ time, and we've already seen that the latter would lead to new circuit lower bounds! Putting all this together, Abboud et al. [11] recently proved the following striking theorem, which they describe in slogan form as "a polylog shaved is a lower bound made":

Theorem 68 (Abboud et al. [11]) *Suppose that* EditDistance *is solvable in $O(n^2/\log^{1000} n)$ time. Then $\mathsf{NEXP} \not\subset \mathsf{NC}^1$.*

A third question is: how does the proof of Theorem 64 evade the known barriers? Because of the way the algorithm for ACCSat exploits the structure of ACC circuits, we shouldn't be surprised if the proof evades the relativization and algebrization barriers. And indeed, using the techniques of Wilson [267] and of Aaronson and Wigderson [10], one can easily construct an oracle A such that $\mathsf{NEXP}^A \subset \mathsf{ACC}^A$, and even an algebraic oracle $\widetilde{A}$ such that $\mathsf{NEXP}^{\widetilde{A}} \subset \mathsf{ACC}^{\widetilde{A}}$, thereby showing that $\mathsf{NEXP} \not\subset \mathsf{ACC}$ is non-relativizing and non-algebrizing. Meanwhile, because it uses diagonalization (in the form of the Nondeterministic Time Hierarchy Theorem), we might say that the proof of Theorem 64 has the "capacity" to evade natural proofs. On the other hand, as I alluded to in Section 6.2.5, it's not yet clear whether ACC is powerful enough to compute pseudorandom functions—and thus, whether it even *has* a natural proofs barrier to evade! The most we can say is that *if* ACC has a natural proofs barrier, *then* Theorem 64 evades it.

Given everything we saw in the previous sections, a final question arises: is there some fourth barrier, beyond relativization, algebrization, and natural proofs, which will inherently prevent even Williams's techniques from proving $\mathsf{P} \neq \mathsf{NP}$, or even (say) $\mathsf{NEXP} \not\subset \mathsf{TC}^0$? One reasonable answer is that this question is premature: in order to identify the barriers to a given set of techniques, we first need to know formally what the techniques *are*—i.e., what properties all the theorems using those techniques have in common—but we can't know that until the techniques have had a decade or more to solidify, and there are at least three or four successful examples of their use. Of course, one obvious "barrier" is that, while Theorem 67 shows that we could get much further just from

[62] This strengthens a previous result of Impagliazzo, Kabanets, and Wigderson [124], who showed that, if such a deterministic algorithm exists that runs in $2^{n^{o(1)}}$ time, then $\mathsf{NEXP} \not\subset \mathsf{P/poly}$.

plausible derandomization assumptions, eventually we might find ourselves asking for faster-than-brute-force algorithms that simply don't exist—in which case, ironic complexity theory would've run out of the irony that it needs as fuel.

Stepping back, I see Theorem 64 as having contributed something important to the quest to prove $\mathsf{P} \neq \mathsf{NP}$, by demonstrating just how much nontrivial work can get done, and how many barriers can be overcome, along the way to applying a 1960s-style hierarchy theorem. Williams's result makes it possible to imagine that, in the far future, $\mathsf{P} \neq \mathsf{NP}$ might be proved by assuming the opposite, then deriving stranger and stranger consequences using thousands of pages of mathematics barely comprehensible to anyone alive today—and yet still, the *coup de grâce* will be a diagonalization, barely different from what Turing did in 1936.

6.5 Arithmetic Complexity Theory

Besides Turing machines and Boolean circuits acting on bits, there's another kind of computation that has enormous relevance to the attempt to prove $\mathsf{P} \neq \mathsf{NP}$. Namely, we can consider computer programs that operate directly on elements of a field, such as the reals or complex numbers. Perhaps the easiest way to do this is via *arithmetic circuits*, which take as input a collection of elements $x_1, \ldots, x_n$ of a field $\mathbb{F}$,[63] and whose operations consist of adding or multiplying any two previous elements—or any previous element and any scalar from $\mathbb{F}$—to produce a new $\mathbb{F}$-element. We then consider the minimum number of operations needed to compute some polynomial $g : \mathbb{F}^n \rightarrow \mathbb{F}$, as a function of n. For concreteness, we can think of $\mathbb{F}$ as the reals $\mathbb{R}$, although we're most interested in algorithms that work over any $\mathbb{F}$. Note that, if we work over a finite field $\mathbb{F}_m$, then we need to specify whether we want to compute g *as a formal polynomial*, or merely as a function over $\mathbb{F}_m$.[64]

At first glance, arithmetic circuits seem more powerful than Boolean circuits, because they have no limit of finite precision: for example, an arithmetic circuit could multiply π and e in a single time step. From another angle, however, arithmetic circuits are weaker, because they have no facility (for example) to extract individual bits from the binary representations of the $\mathbb{F}$ elements: they can *only* manipulate them as $\mathbb{F}$ elements. In general, the most we can say is that, *if* an input has helpfully been encoded using the elements $0, 1 \in \mathbb{F}$ only, *then* an arithmetic circuit can simulate a Boolean one, by using $x \rightarrow 1 - x$ to simulate NOT, multiplication to simulate Boolean AND, and so on. But for arbitrary inputs, such a simulation might be impossible.

Thus, arithmetic circuits represent a different kind of computation: or rather, a generalization of the usual kind, since we can recover ordinary Boolean computation by setting $\mathbb{F} = \mathbb{F}_2$. A major reason to focus on arithmetic circuits is that it often seems easier—or better, less absurdly hard!—to understand circuit size in the arithmetic setting than in the Boolean one. The usual explanation given for this is the so-called "yellow books argument": arithmetic complexity brings us closer to continuous mathematics, about which we have centuries' worth of deep knowledge (e.g., algebraic geometry and representation theory) that's harder to apply in the Boolean case.

One remark: in the rest of the section, I'll talk exclusively about arithmetic *circuit* complexity: that is, about nonuniform arithmetic computations, and the arithmetic analogues of questions such as NP versus $\mathsf{P/poly}$ (see Section 5.2). But it's also possible to develop a theory of *arithmetic Turing machines*, which (roughly speaking) are like arithmetic circuits except that they're uniform, and therefore need loops, conditionals, memory registers, and so on. See the book of Blum, Cucker,

[63] I'll restrict to fields here for simplicity, but one can also consider (e.g.) rings.

[64] To illustrate, 0 and $2x$ are equal as functions over the finite field $\mathbb{F}_2$, but not equal as formal polynomials.

Shub, and Smale (BCSS) [49] for a beautiful exposition of this theory. In the BCSS framework, one can ask precise analogues of the $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ question for Turing machines over arbitrary fields $\mathbb{F}$, such as $\mathbb{R}$ or $\mathbb{C}$, recovering the "ordinary, Boolean" $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ question precisely when $\mathbb{F}$ is finite. At present, no implications are known among the $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$, the $\mathsf{P}_{\mathbb{R}} \stackrel{?}{=} \mathsf{NP}_{\mathbb{R}}$, and the $\mathsf{P}_{\mathbb{C}} \stackrel{?}{=} \mathsf{NP}_{\mathbb{C}}$ questions, although it's known that $\mathsf{P}_{\mathbb{C}} \neq \mathsf{NP}_{\mathbb{C}}$ implies $\mathsf{NP} \not\subset \mathsf{P/poly}$ (see for example Bürgisser [61, Chapter 8]).[65]

The problems of proving $\mathsf{P}_{\mathbb{R}} \neq \mathsf{NP}_{\mathbb{R}}$ and $\mathsf{P}_{\mathbb{C}} \neq \mathsf{NP}_{\mathbb{C}}$ are known to be closely related to the problem of proving arithmetic circuit lower bounds, which we'll discuss in the following sections. I can't resist giving one example of a connection, due to BCSS [49]. Given a positive integer n, let $\tau(n)$ be the number of operations in the smallest arithmetic circuit that takes the constant 1 as its sole input, and that computes n using additions, subtractions, and multiplications. For example, we have

- $\tau(2) = 1$ via $1+1$,
- $\tau(3) = 2$ via $1+1+1$,
- $\tau(4) = 2$ via $(1+1)^2$, ...

Also, let $\tau^*(n)$ be the minimum of $\tau(kn)$ over all positive integers k.

Theorem 69 (BCSS [49]) *Suppose $\tau^*(n!)$ grows faster than $(\log n)^{O(1)}$. Then $\mathsf{P}_{\mathbb{C}} \neq \mathsf{NP}_{\mathbb{C}}$.*[66]

6.5.1 Permanent Versus Determinant

Just as $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ is the "flagship problem" of Boolean complexity theory, so the central, flagship problem of arithmetic complexity is that of *permanent versus determinant*. This problem concerns the following two functions of an $n \times n$ matrix $X \in \mathbb{F}^{n \times n}$:

$$\mathrm{Per}(X) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} x_{i,\sigma(i)},$$

$$\mathrm{Det}(X) = \sum_{\sigma \in S_n} (-1)^{\mathrm{sgn}(\sigma)} \prod_{i=1}^{n} x_{i,\sigma(i)}.$$

Despite the similarity of their definitions—they're identical apart from the $(-1)^{\mathrm{sgn}(\sigma)}$—the permanent and determinant have dramatic differences. The determinant is computable in polynomial time, for example by using Gaussian elimination. (Indeed, the determinant is computable in $O(n^\omega)$ time, where $\omega \in [2, 2.373]$ is the matrix multiplication exponent; see Section 4.) The determinant has many other interpretations—for example, the product of X's eigenvalues, and the volume of the parallelepiped spanned by its row vectors—giving it a central role in linear algebra and geometry.

By contrast, Valiant [247] proved in 1979 that the permanent is #P-complete. Thus, a polynomial-time algorithm for the permanent would imply even more than $\mathsf{P} = \mathsf{NP}$: it would

[65] The central difference between the $\mathsf{P}_{\mathbb{R}} \stackrel{?}{=} \mathsf{NP}_{\mathbb{R}}$ and $\mathsf{P}_{\mathbb{C}} \stackrel{?}{=} \mathsf{NP}_{\mathbb{C}}$ questions is simply that, because $\mathbb{R}$ is an ordered field, one defines Turing machines over $\mathbb{R}$ to allow comparisons $(<, \leq)$ and branching on their results.

[66] In later work, Bürgisser [63] showed that the same conjecture about $\tau^*(n!)$ (called the *τ-conjecture*) would also imply Valiant's Conjecture 70, that the permanent has no polynomial-size arithmetic circuits.

yield an efficient algorithm not merely to solve NP-complete problems, but to count how many solutions they have. In some sense, the #P-completeness of the permanent helps to *explains* why Per, unlike Det, has no simple geometric or linear-algebraic interpretations: if such interpretations existed, then they might imply $\mathsf{P} = \mathsf{P}^{\#\mathsf{P}}$.

In the arithmetic model, there exist circuits of size $O\left(n^3\right)$, and even size $O\left(n^\omega\right)$, that compute $\mathrm{Det}\left(X\right)$ as a formal polynomial in the entries of X, and that work over an arbitrary field $\mathbb{F}$. By contrast, Valiant conjectured the following.

Conjecture 70 (Valiant's Conjecture) *Any arithmetic circuit for* $\mathrm{Per}\left(X\right)$ *requires size super-polynomial in* n*, over any field of characteristic other than* 2.[67,68]

Bürgisser [62] showed that, if Conjecture 70 fails over any field of positive characteristic, or if it fails over any field of characteristic zero *and the Generalized Riemann Hypothesis holds*, then $\mathsf{P}^{\#\mathsf{P}} \subset \mathsf{P/poly}$, and hence $\mathsf{NP} \subset \mathsf{P/poly}$.[69] (The main difficulty in proving this result is just that an arithmetic circuit might have very large constants hardwired into it.) On the other hand, no converses to this result are currently known. It's conceivable, for example, that we could have $\mathsf{P} = \mathsf{P}^{\#\mathsf{P}}$ for some "inherently Boolean" reason, even if the permanent required arithmetic circuits of exponential size. To put it another way, Conjecture 70 could serve as an "arithmetic warmup"—some would even say an "arithmetic prerequisite"—to Boolean separations such as $\mathsf{P}^{\#\mathsf{P}} \not\subset \mathsf{P/poly}$ and $\mathsf{P} \neq \mathsf{NP}$.

Better yet, Conjecture 70 turns out to be implied by (and nearly equivalent to) an appealing mathematical conjecture, which makes no direct reference to computation or circuits. Let's say that the $n \times n$ permanent *linearly embeds* into the $m \times m$ determinant, if it's possible to express $\mathrm{Per}\left(X\right)$ (for an $n \times n$ matrix $X \in \mathbb{F}^{n \times n}$) as $\mathrm{Det}\left(L\left(X\right)\right)$, where $L\left(X\right)$ is an $m \times m$ matrix each of whose entries is an affine combination of the entries of X. Then let $D\left(n\right)$ be the smallest m such that the $n \times n$ permanent linearly embeds into the $m \times m$ determinant.

Grenet [102] proved the following:

Theorem 71 (Grenet [102]) $D\left(n\right) \leq 2^n - 1$.

To illustrate, when $n = 3$ we have

$$\mathrm{Per}\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} = \mathrm{Det}\begin{pmatrix} 0 & a & d & g & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & i & f & 0 \\ 0 & 0 & 1 & 0 & 0 & c & i \\ 0 & 0 & 0 & 1 & c & 0 & f \\ e & 0 & 0 & 0 & 1 & 0 & 0 \\ h & 0 & 0 & 0 & 0 & 1 & 0 \\ b & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

By contrast, the best current lower bound on $D\left(n\right)$ is quadratic, and was proved by Mignon and Ressayre [170] in 2004, following a long sequence of linear lower bounds:

[67] Fields of characteristic 2, such as $\mathbb{F}_2$, are a special case: there, the permanent and determinant are *equivalent*, so in particular $\mathrm{Per}\left(X\right)$ has polynomial-size arithmetic circuits.

[68] In the literature, Conjecture 70 is often called the $\mathsf{VP} \neq \mathsf{VNP}$ conjecture, with VP and VNP being arithmetic analogues of P and NP respectively. I won't use that terminology in this survey, for several reasons: (1) VP is arguably more analogous to NC than to P, (2) VNP is arguably more analogous to #P than to NP, and (3) Conjecture 70 is almost always studied as a nonuniform conjecture, more analogous to $\mathsf{NP} \not\subset \mathsf{P/poly}$ than to $\mathsf{P} \neq \mathsf{NP}$.

[69] Indeed, #P would even have polynomial-size circuits of depth $\log^{O(1)} n$.

Theorem 72 (Mignon and Ressayre [170]) *$D(n) \geq n^2/2$.*

(Actually, Mignon and Ressayre proved Theorem 72 only for fields of characteristic 0. Their result was then extended to all fields of characteristic other than 2 by Cai, Chen, and Li [68] in 2008.)

The basic idea of the proof of Theorem 72 is to consider the *Hessian matrix* of a polynomial $p : \mathbb{F}^N \rightarrow \mathbb{F}$, or the matrix of second partial derivatives, evaluated at some particular point $X_0 \in \mathbb{F}^N$:

$$H_p(X_0) := \begin{pmatrix} \frac{\partial^2 p}{\partial x_1^2}(X_0) & \cdots & \frac{\partial^2 p}{\partial x_1 \partial x_N}(X_0) \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 p}{\partial x_N \partial x_1}(X_0) & \cdots & \frac{\partial^2 p}{\partial x_N^2}(X_0) \end{pmatrix}.$$

Here we mean the "formal" partial derivatives of p: even if $\mathbb{F}$ is a finite field, we can still symbolically differentiate a polynomial over $\mathbb{F}$, to produce new polynomials over smaller sets of variables. In general, when we're trying to lower-bound the difficulty of computing a polynomial p, a common technique in arithmetic complexity is to look at various partial derivatives $\frac{\partial^k p}{\partial x_{i_1} \cdots \partial x_{i_k}}$—and in particular, at the dimensions of vector spaces spanned by those partial derivatives, or the ranks of matrices formed from them—and then argue that, if p had a small circuit (or formula, or whatever), then those dimensions or ranks couldn't possibly be as high as they are.

In the case of Theorem 72, we prove the following two statements:

(1) If p is the permanent, of an $n \times n$ matrix of $N = n^2$ indeterminates, then there exists a point $X_0 \in \mathbb{F}^N$ such that $\text{rank}(H_p(X_0)) = N$.

(2) If p is the determinant of an $m \times m$ matrix of affine functions in the N indeterminates, then $\text{rank}(H_p(X)) \leq 2m$ for every X.

Combining these, we get $m \geq n^2/2$, if p is both the $n \times n$ permanent and an $m \times m$ determinant.

So to summarize, the "blowup" $D(n)$ in embedding the permanent into the determinant is known to be at least quadratic and at most exponential. The huge gap here becomes a bit less surprising, once we know that $D(n)$ is tightly connected to the arithmetic circuit complexity of the permanent. In particular, recall that a *formula* is just a circuit in which every gate has a fanout of 1. Then Valiant [246] showed the following:

Theorem 73 (Valiant [246]) *$D(n) \leq F(n)+1$, where $F(n)$ is the size of the smallest arithmetic formula for the $n \times n$ permanent.*

Thus, if we could prove that $D(n)$ grew faster than any polynomial, we'd have shown that the permanent has no polynomial-size formulas. But heightening the interest still further, Valiant et al. [249] showed that in the arithmetic world, there's a surprisingly tight connection between formulas and circuits:

Theorem 74 (Valiant et al. [249]) *If a degree-d polynomial has an arithmetic circuit of size s, then it also has an arithmetic formula of size $(sd)^{O(\log d)}$.*

Theorem 74 implies that $D(n) \leq C(n)^{O(\log n)}$, where $C(n)$ is the size of the smallest arithmetic *circuit* for the $n \times n$ permanent. This means that, if we could prove that $D(n)$ grew not only superpolynomially but faster than $n^{O(\log n)}$, we'd also have shown that $C(n)$ grew superpolynomially, thereby establishing Valiant's Conjecture 70.

But lower-bounding $D(n)$ is not merely sufficient for proving Valiant's Conjecture; it's also necessary! For recall that the $n \times n$ determinant has an arithmetic circuit of size $O\left(n^3\right)$, and even $O\left(n^{\omega}\right)$. So we get the following chain of implications:

$$\begin{aligned} D(n) > n^{O(\log n)} &\implies F(n) > n^{O(\log n)} \text{ (by Theorem 73)} \\ &\implies C(n) > n^{O(1)} \text{ (by Theorem 74; this is Valiant's Conjecture 70)} \\ &\implies D(n) > n^{O(1)} \text{ (by the } n^{O(1)} \text{ arithmetic circuit for determinant)} \\ &\implies F(n) > n^{O(1)} \text{ (by Theorem 73).} \end{aligned}$$

Today, a large fraction of the research aimed at proving $\mathsf{P} \neq \mathsf{NP}$ is aimed, more immediately, at proving Valiant's Conjecture 70 (see Agrawal [14] for a survey focusing on that goal). The hope is that, on the one hand, powerful tools from algebraic geometry and other fields can be brought to bear on Valiant's problem, but on the other, that solving it could provide insight about the original $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ problem.

6.5.2 Arithmetic Circuit Lower Bounds

I won't do justice in this survey to the now-impressive body of work motivated by Conjecture 70; in particular, I'll say little about proof techniques. Readers who want to learn more about arithmetic circuit lower bounds should consult Shpilka and Yehudayoff [225, Chapter 3] for an excellent survey circa 2010, or Saraf [216] for a 2014 update. Briefly, though, computer scientists have tried to approach Conjecture 70 much as they've approached $\mathsf{NP} \not\subset \mathsf{P/poly}$, by proving lower bounds against more and more powerful arithmetic circuit classes. In that quest, they've had some notable successes (paralleling the Boolean successes), but have also run up against some major differences from the Boolean case.

For starters, just as Razborov [206] and others considered monotone Boolean circuits, one can also consider *monotone arithmetic circuits* (over fields such as $\mathbb{R}$ or $\mathbb{Q}$), in which all coefficients need to be positive. Since the determinant involves -1 coefficients, it doesn't make sense to ask about monotone circuits for $\mathrm{Det}(X)$, but one can certainly ask about the monotone circuit complexity of $\mathrm{Per}(X)$. And already in 1982, Jerrum and Snir [128] proved the following arithmetic counterpart of Razborov's Theorem 46:

Theorem 75 (Jerrum and Snir [128]) *Any monotone circuit for* $\mathrm{Per}(X)$ *requires size* $2^{\Omega(n)}$.

As another example, just as computer scientists considered constant-depth Boolean circuits (the classes AC^0, ACC, TC^0, and so on), so we can also consider *constant-depth arithmetic circuits*, which are conventionally denoted $\Sigma\Pi$, $\Sigma\Pi\Sigma$, etc. to indicate whether they represent a multivariate polynomial as a sum of products, a sum of product of sums, etc. It's trivial to prove exponential lower bounds on the sizes of depth-two ($\Sigma\Pi$) circuits: that just amounts to lower-bounding the number of monomials in a polynomial. More interesting is the following result:

Theorem 76 (Grigoriev and Karpinski [103], Grigoriev and Razborov [104]) *Over a finite field, any* $\Sigma\Pi\Sigma$ *circuit for* $\mathrm{Det}(X)$ *requires size* $2^{\Omega(n)}$. *(Indeed, this is true even for circuits representing* $\mathrm{Det}(X)$ *as a function.)*

Curiously, over *infinite* fields, the best lower bound that we have for the determinant is still a much weaker one, due to Shpilka and Wigderson [224]:

Theorem 77 (Shpilka and Wigderson [224]) *Over infinite fields, any* $\Sigma\Pi\Sigma$ *circuit for* $\mathrm{Det}(X)$ *requires size* $\Omega\left(n^4/\log n\right)$.[70]

Theorems 76 and 77 are stated for the determinant, although they have analogues for the permanent. In any case, these results certainly don't succeed in showing that the permanent is *harder* than the determinant.

The situation is better when we restrict the fanin of the multiplication gates. In particular, by a $\Sigma\Pi^{[a]}\Sigma\Pi^{[b]}$ circuit, let's mean a depth-4 circuit where every inner multiplication gate has fanin at most a, and every bottom multiplication gate has fanin at most b. Then in 2013, Gupta et al. [109] proved the following.

Theorem 78 (Gupta, Kamath, Kayal, and Saptharishi [109]) *Any* $\Sigma\Pi^{[O(\sqrt{n})]}\Sigma\Pi^{[\sqrt{n}]}$ *circuit for* $\mathrm{Per}(X)$ *or* $\mathrm{Det}(X)$ *requires size* $2^{\Omega(\sqrt{n})}$.

Subsequently, Kayal, Saha, and Saptharishi [138] proved a size lower bound of $n^{\Omega(\sqrt{n})}$ for such circuits, though not for the permanent or determinant but for a different explicit polynomial.

The situation is also better when we restrict to *homogeneous* arithmetic circuits. These are circuits where every gate is required to compute a homogeneous polynomial: that is, one where all the monomials have the same degree. Here Nisan and Wigderson [191] established the following in 1997.

Theorem 79 (Nisan and Wigderson [191]) *Over any field, any homogeneous* $\Sigma\Pi\Sigma$ *circuit for* $\mathrm{Det}(X)$ *requires size* $2^{\Omega(n)}$.

Going further, in 2014 Kayal, Limaye, Saha, and Srinivasan [137] gave an explicit polynomial in n variables for which any homogeneous $\Sigma\Pi\Sigma\Pi$ circuit requires size $n^{\Omega(\sqrt{n})}$ (compared to Theorem 79's $2^{\Omega(\sqrt{n})}$, as a function of the number of input variables).

It's natural to wonder: why are we stuck talking about depth-3 and depth-4 arithmetic circuits? Why couldn't we show that the permanent and determinant have no *constant-depth* arithmetic circuits of subexponential size, just like Theorem 49 and its successors showed that Parity has no constant-depth Boolean circuits of subexponential size? After all, wasn't the whole point of arithmetic complexity that it was supposed to be *easier* than Boolean complexity?

In 2008, Agrawal and Vinay [16] gave a striking answer to these questions; they called their answer "the chasm at depth four." In particular, building on the earlier work of Valiant et al. [249] (Theorem 74), Agrawal and Vinay showed that, if we managed to prove strong enough lower bounds for depth-4 arithmetic circuits, then we'd also get superpolynomial lower bounds for *arbitrary* arithmetic circuits! Here's one special case of their result:

[70] As this survey was being written, Kayal, Saha, and Tavenas [139] announced a proof that a certain explicit polynomial, albeit not the permanent or determinant, requires $\Sigma\Pi\Sigma$ circuits of size $\Omega\left(n^3/\log^2 n\right)$, over any field $\mathbb{F}$. By comparison, as a function of the number of input variables (n^2), Shpilka and Wigderson's $\Omega\left(n^4/\log n\right)$ lower bound for the determinant [224] is "only" quadratic.

Theorem 80 (Agrawal and Vinay [16]) *Suppose that* $\mathrm{Per}\,(X)$ *requires depth-4 arithmetic circuits (even homogeneous ones) of size* $2^{\Omega(n)}$. *Then* $\mathrm{Per}\,(X)$ *requires arithmetic circuits of superpolynomial size, and Valiant's Conjecture 70 holds.*

Subsequently, Koiran [147] and Tavenas [242] showed that Valiant's Conjecture would follow, not merely from a $2^{\Omega(n)}$ size lower bound for homogeneous depth-4 circuits computing the permanent, but from *any* size lower bound better than $n^{\Omega(\sqrt{n})}$. In an even more exciting development, Gupta et al. [108] reduced the depth from four to three (though only for fields of characteristic 0, and no longer allowing homogeneity):

Theorem 81 (Gupta, Kamath, Kayal, and Saptharishi [108]) *Suppose that* $\mathrm{Per}\,(X)$ *requires depth-3 arithmetic circuits of size more than* $n^{\Omega(\sqrt{n})}$, *over fields of characteristic* 0. *Then* $\mathrm{Per}\,(X)$ *requires arithmetic circuits of superpolynomial size, and Valiant's Conjecture 70 holds.*

These results can be considered extreme versions of the depth reduction of Brent [56] (see Proposition 26). I should mention that all of these results hold, not just for the permanent, but for *any* homogeneous polynomial of degree $n^{O(1)}$. In particular, by applying their depth reduction "in the opposite direction" for the determinant, Gupta et al. [108] were able to show that there *exist* depth-3 arithmetic circuits of size $n^{O(\sqrt{n})}$ for $\mathrm{Det}\,(X)$. This provides an interesting counterpoint to the result of Nisan and Wigderson [191] (Theorem 79), which showed that $2^{\Omega(n)}$ gates are needed for the determinant if we restrict to depth-3 *homogeneous* circuits.

There are yet other results in this vein, which give yet other tradeoffs. But perhaps we should step back from the flurry of theorems and try to summarize. After decades of research in arithmetic circuit complexity, we now have lower bounds of the form $n^{\Omega(\sqrt{n})}$ on the sizes of depth-3 and depth-4 arithmetic circuits computing explicit polynomials (subject to various technical restrictions). On the other hand, we also have a deep explanation for why the progress has stopped at the specific bound $n^{\Omega(\sqrt{n})}$: because *any lower bound even slightly better than that would already prove Valiant's Conjecture, that the permanent is superpolynomially harder than the determinant!* It's as if, in arithmetic complexity, we reach a terrifying precipice—beyond which we can no longer walk but need to fly—sooner than we do in the Boolean case. And around 2014, we learned exactly where that precipice is and walked right up to it, but we still haven't jumped.[71,72]

In this connection, it's worth pointing out that, with the exception of Theorem 72 by Mignon and Ressayre [170], none of the results in this section actually *differentiate* the permanent from the determinant: that is, none of them prove a lower bound for $\mathrm{Per}\,(X)$ better than the analogous lower bound known for $\mathrm{Det}\,(X)$. Eventually, of course, any proof of Valiant's Conjecture *will* need to explain why the permanent is harder than the determinant, which is one of the main motivations for the Mulmuley-Sohoni program (see Section 6.6).

Let me end this section by discussing two striking results of Ran Raz, and one of Pascal Koiran, that didn't quite fit into the narrative above. The first result is a superpolynomial lower bound on the sizes of *multilinear formulas*. An arithmetic formula is called *multilinear* if the polynomial computed by each gate is a multilinear polynomial (that is, no variable is raised to a higher power

[71] Or perhaps, we've jumped many times, but each time hit the bottom rather than flew!

[72] As this survey was being written, Forbes, Kumar, and Saptharishi [84] gave yet another interesting result sharpening the contours of the "chasm at depth four": namely, they showed that lower bounds on homogeneous depth-4 arithmetic circuits to compute *Boolean functions* (rather than formal polynomials), and which are only "slightly" stronger than lower bounds that have already been shown, would imply a separation between #P and ACC.

than 1). Notice that the permanent and determinant are both multilinear polynomials. For that reason, they can be computed by multilinear formulas, and it makes sense to ask about the size of the smallest such formulas.

In a 2004 breakthrough, Raz [201] proved the following.

Theorem 82 (Raz [201]) *Any multilinear formula for* $\mathrm{Per}(X)$ *or* $\mathrm{Det}(X)$ *requires size* $n^{\Omega(\log n)}$.[73]

What made Theorem 82 striking was that there was no restriction on the formula's depth. The proof was via the random restriction method from Section 6.2.3, combined with the idea (common in arithmetic complexity) of using matrix rank as a progress measure. In more detail, let $p : \{0,1\}^n \rightarrow \mathbb{R}$ be a polynomial computed by a small multilinear formula: for simplicity, we'll take p's inputs to be Boolean. Then basically, we randomly partition p's input variables into two small sets $X = \{x_1, \ldots, x_k\}$ and $Y = \{y_1, \ldots, y_k\}$, and a large set Z of size $n - 2k$. (Here we should imagine, say, $k = n^{1/3}$.) We then randomly fix the variables in Z to 0's or 1's, while leaving the variables in X and Y unfixed. Next, we define a matrix $M \in \mathbb{R}^{2^k \times 2^k}$, whose rows are indexed by the 2^k possible assignments to X, whose columns are indexed by the 2^k possible assignments to Y, and whose (X, Y) entry equals $p(X, Y, Z)$. Finally, we prove the following two statements:

- With high probability, M has rank much smaller than 2^k. This is the hard part of the proof: we use the assumption that p has a small multilinear formula, and then argue by induction on the formula.
- If p represents the function f of interest to us (say, the permanent or determinant), then $\mathrm{rank}(M) = 2^k$ with certainty.

Together, these yield the desired contradiction, showing that f can't have had a small multilinear formula after all.

It seems likely that the lower bound in Theorem 82 could be improved from $n^{\Omega(\log n)}$ all the way up to $2^{\Omega(n)}$, but this remains open. Raz and Yehudayoff [204] did manage to prove an exponential lower bound for *constant-depth* multilinear formulas computing the permanent or determinant; and in a separate work [205], they also proved a $2^{\Omega(n)}$ lower bound for "non-cancelling" multilinear formulas computing an explicit polynomial f (not the permanent or determinant). Here "non-cancelling"—a notion that I defined in [2]—basically means that nowhere in the formula are we allowed to add two polynomials that "almost perfectly" cancel each other out, leaving only a tiny residue.

Of course, just like with the arithmetic circuit lower bounds discussed earlier, so far all the known multilinear formula lower bounds fail to distinguish the permanent from the determinant.

The second result of Raz's concerns so-called *elusive functions*. Given a polynomial curve $f : \mathbb{C} \rightarrow \mathbb{C}^n$, Raz calls f *elusive* if f is not contained in the image of any polynomial mapping $g : \mathbb{C}^{n-1} \rightarrow \mathbb{C}^n$ of degree 2. He then proves the following beautiful theorem.

Theorem 83 (Raz [202]) *Suppose there exists an elusive function whose coefficients can be computed in polynomial time. Then* $\mathrm{Per}(X)$ *requires arithmetic circuits of superpolynomial size, and Valiant's Conjecture 70 holds.*

[73] An immediate corollary is that any multilinear *circuit* for $\mathrm{Per}(X)$ or $\mathrm{Det}(X)$ requires *depth* $\Omega\left(\log^2 n\right)$.

Arguably, this makes Valiant's Conjecture look *even more* like a question of pure algebraic geometry than it did before! As evidence that the "elusive function" approach to circuit lower bounds is viable, Raz then constructs an explicit f that's elusive in a weak sense, which is already enough to imply the following new lower bound:

Theorem 84 (Raz [202]) *For every r, there is an explicit polynomial p with n variables and degree $O(r)$, such that any depth-r arithmetic circuit for p (over any field) requires size $n^{1+\Omega(1/r)}$.*

Now for the result of Koiran.

Theorem 85 (Koiran [146]) *Suppose that any univariate real polynomial, of the form*

$$p(x) = \sum_{i=1}^{\ell} \prod_{j=1}^{m} \sum_{k=1}^{n} a_{ijk} x^{e_{ijk}},$$

has at most $(\ell mn)^{O(1)}$ real zeroes. Then $\mathrm{Per}(X)$ requires arithmetic formulas of superpolynomial size (and indeed, $D(n) > n^{O(1)}$).

In fact it would suffice to upper-bound the number of *integer* zeroes of such a polynomial by $(\ell mn)^{O(1)}$. Note that, if we had asked about *complex* zeroes, then Theorem 85 would badly fail, because of counterexamples such as $x^{2^n} - 1$. But with real zeroes, no counterexample is known, and Theorem 85 once again raises the tantalizing possibility that tools from analysis could be brought to bear on the permanent versus determinant problem.

6.5.3 Arithmetic Natural Proofs?

In Section 6.5.2, we saw arithmetic circuit lower bounds that, again and again, seem to go "right up to the brink" of proving Valiant's Conjecture, but then stop short. Given this, it's natural to wonder what the barriers are to further progress in arithmetic complexity, and how they relate to the barriers in the Boolean case.

We've already discussed one obvious barrier, which is that eventually we need techniques that work for the permanent but *fail* for the determinant. It might also be interesting to define an arithmetic analogue of the relativization barrier (Section 6.1.2). To my knowledge, this hasn't been done, but my guess is that in the arithmetic setting, the natural choices for oracles would look a lot like the algebraic oracles studied by Aaronson and Wigderson [10] (see Section 6.3.3). With a notion of "oracle" in hand, one could probably show that most arithmetic circuit lower bounds require arithmetically non-relativizing techniques. On the other hand, this wouldn't be much of an obstruction, since even the results discussed in Section 6.5.2 should *already* evade the relativization barrier, for the same reason as those of Sections 6.2.3 and 6.2.4.

In the rest of this section, I'd like to discuss the contentious question of whether or not arithmetic circuit complexity faces a natural proofs barrier, in the sense of Razborov and Rudich [210]. Recall from Section 6.2.5 that a circuit lower bound proof is called *natural* if, besides proving that the specific function f of interest to us is not in a circuit class $\mathcal{C}$, the proof also provides a *polynomial-time algorithm* A that takes as input a function's truth table, and that certifies a $1/n^{O(1)}$ fraction of all functions as not belonging to $\mathcal{C}$. Such an A can be used to distinguish functions in $\mathcal{C}$ from random functions with non-negligible bias. Meanwhile, the class $\mathcal{C}$ has a *natural proofs barrier* if $\mathcal{C}$

contains *pseudorandom function families*, which can't be so distinguished from random functions, and whose existence is therefore incompatible with the existence of A.

In the arithmetic setting, presumably we'd call a proof *natural* if it yields a polynomial-time algorithm[74] that takes as input, say, the complete output table of a homogeneous degree-d polynomial $p : \mathbb{F}^n \rightarrow \mathbb{F}$ over a finite field $\mathbb{F}$, and that certifies a $1/n^{O(1)}$ fraction of all such polynomials as not belonging to the arithmetic circuit class $\mathcal{C}$. Also, we'd say that $\mathcal{C}$ has a *natural proofs barrier* if $\mathcal{C}$ contains *pseudorandom polynomial families*. By this, we mean families of homogeneous degree-d polynomials, $p_s : \mathbb{F}^n \rightarrow \mathbb{F}$, that no $|\mathbb{F}|^{O(n)}$-time algorithm can distinguish from uniformly-random homogeneous degree-d polynomials with non-negligible bias. (We can no longer talk about uniformly-random *functions*, since an algorithm can easily ascertain, for example, that p_s is a degree-d polynomial.) By exactly the same logic as in the Boolean case, if $\mathcal{C}$ is powerful enough to compute pseudorandom polynomials, then no natural proof can show that a polynomial isn't in $\mathcal{C}$.

Now, one point that's *not* disputed is that all the arithmetic circuit lower bounds discussed in Section 6.5.2 are natural in the above sense. I didn't say much about how the lower bounds are proved, but as mentioned in Section 6.5.1, arithmetic circuit lower bounds generally proceed by finding some parameter $\alpha(p)$ associated with a polynomial p—say, the rank of its Hessian matrix, or the dimension of a vector space spanned by p's partial derivatives—to use as a "progress measure." The proof then argues that

(1) $\alpha(p)$ is large for the specific polynomial p of interest to us (say, the permanent or determinant), but

(2) every gate added to our circuit or formula can only increase $\alpha(p)$ by so much,

thereby implying that p requires many gates. Furthermore, virtually any progress measure α that's a plausible choice for such an argument—and certainly the ones used in the existing results—will be computable in $|\mathbb{F}|^{O(n)}$ time, and will be maximized by a *random* polynomial p of the appropriate degree. Alas, this implies that the argument is natural! If the circuit class $\mathcal{C}$ has a natural proofs barrier, then no such argument can possibly prove $p \notin \mathcal{C}$.

The part that's controversial is whether arithmetic circuit classes *do* have a natural proofs barrier. To show that they did, we'd need plausible candidates for pseudorandom polynomials—say, homogeneous degree-d polynomials $p : \mathbb{F}^n \rightarrow \mathbb{F}$ that actually have small arithmetic circuits, but that look to any efficient test just like random homogeneous polynomials of degree d. The trouble is that, while cryptographers know a great deal about how to construct pseudorandom functions, the accepted constructions are all "inherently Boolean"; they don't work in the setting of low-degree polynomials over a finite field.

Thus, to take one example, the work of Goldreich, Goldwasser, and Micali (GGM) [98], combined with that of Håstad et al. [116], shows how to build a pseudorandom function family starting from any *one-way function* (see Section 5.3.1). And indeed, Razborov and Rudich [210] used a variant of the GGM construction in their original paper on natural proofs. However, if we try to implement the GGM construction using arithmetic circuits—say, using multiplication for the AND gates, $1-x$ for the NOT gates, etc.—we'll find that we've produced an arithmetic circuit of $n^{O(1)}$ depth, which computes a polynomial of $\exp\left(n^{O(1)}\right)$ degree: far too large.

[74] For simplicity, here I'll assume that we mean an "ordinary" (Boolean) polynomial-time algorithm, though one could also require polynomial-time algorithms in the arithmetic model.

As I mentioned in Section 6.2.5, if we're willing to assume the hardness of specific cryptographic problems, then there are also much more direct constructions of pseudorandom functions, which produce circuits of much lower depth. In particular, there's the construction of Naor and Reingold [187], which is based on factoring and discrete logarithm; and that of Banerjee et al. [37], which is based on noisy systems of linear equations. Unfortunately, examination of these constructions reveals that they, too, require treating the input as a string of bits rather than of finite field elements. So for example, the Naor-Reingold construction involves modular exponentiation, which of course goes outside the arithmetic circuit model, where only addition and multiplication are allowed.

At this point I can't resist stating my own opinion, which is that the issue here is partly technical but also partly social. Simply put: Naor-Reingold and Banerjee et al. are taken to be relevant to natural proofs, because factoring, discrete logarithm, and solving noisy systems of linear equations have become *accepted by the community of cryptographers* as plausibly hard problems. Since real computers use Boolean circuits, and since in practice one normally needs pseudorandom *functions* rather than polynomials, cryptographers have had extremely little reason to study pseudorandom low-degree polynomials that are computed by small arithmetic circuits over finite fields. If they *had* studied that, though, it seems plausible that they would've found decent candidates for such polynomials, and formed a social consensus that they indeed seem hard to distinguish from random polynomials.

Motivated by that thought, in a 2008 blog post [5], I offered my own candidate for a pseudorandom family of polynomials, $p_s : \mathbb{F}^n \rightarrow \mathbb{F}$, which are homogeneous of degree $d = n^{O(1)}$. My candidate was simply this: motivated by Valiant's result [246] that the determinant can express any arithmetic formula (Theorem 73), take the random seed s to encode d^2 uniformly-random linear functions, $L_{i,j} : \mathbb{F}^n \rightarrow \mathbb{F}$ for all $i, j \in \{1, \ldots, d\}$. Then set

$$p_s(x_1, \ldots, x_n) := \text{Det} \begin{pmatrix} L_{1,1}(x_1, \ldots, x_n) & \cdots & L_{1,d}(x_1, \ldots, x_n) \\ \vdots & \ddots & \vdots \\ L_{d,1}(x_1, \ldots, x_n) & \cdots & L_{d,d}(x_1, \ldots, x_n) \end{pmatrix}.$$

My conjecture is that, at least when d is sufficiently large, a random p_s drawn from this family should require $\exp\left(d^{\Omega(1)}\right)$ time to distinguish from a random homogeneous polynomial of degree d, if we're given the polynomial $p : \mathbb{F}^n \rightarrow \mathbb{F}$ by a table of $|\mathbb{F}|^n$ values. If d is a large enough polynomial in n, then $\exp\left(d^{\Omega(1)}\right)$ is greater than $|\mathbb{F}|^{O(n)}$, so the natural proofs barrier would apply.

So far there's been little study of this conjecture. Neeraj Kayal and Joshua Grochow (personal communication) have pointed out to me that the Mignon-Ressayre Theorem (Theorem 72) implies that p_s can be efficiently distinguished from a uniformly-random degree-d homogeneous polynomial whenever $d < n/2$. For Mignon and Ressayre show that the Hessian matrix, $H_p(X)$, satisfies $\text{rank}(H_p(X)) \leq 2d$ for all X if p is a $d \times d$ determinant, whereas a random p would have $\text{rank}(H_p(X)) = n$ for most X with overwhelming probability, so this provides a distinguisher. However, it's not known what happens for larger d.

Of course, if our goal is to prove $\mathsf{P} \neq \mathsf{NP}$ or $\mathsf{NP} \not\subset \mathsf{P/poly}$, then perhaps the whole question of arithmetic natural proofs is ultimately beside the point. For to prove $\mathsf{NP} \not\subset \mathsf{P/poly}$, we'll need to become experts at overcoming the natural proofs barrier *in any case*: either in the arithmetic world, or if not, then when we move from the arithmetic world back to the Boolean one.

6.6 Geometric Complexity Theory

I'll end this survey with some extremely high-level remarks about Geometric Complexity Theory (GCT): an ambitious program to prove $\mathsf{P} \neq \mathsf{NP}$ and related conjectures using algebraic geometry and representation theory. This program has been pursued since the late 1990s; it was started by Ketan Mulmuley, with important contributions from Milind Sohoni and others. GCT has changed substantially since its inception: for example, as we'll see, recent negative results by Ikenmeyer and Panova [120] among others have decisively killed certain early hopes for the GCT program, while still leaving many avenues to explore in GCT more broadly construed. Confusingly, the term "GCT" can refer either to the original Mulmuley-Sohoni vision, *or* to the broader interplay between complexity and algebraic geometry, which is now studied by a community of researchers, many of whom deviate from Mulmuley and Sohoni on various points. In this section, I'll start by explaining the original Mulmuley-Sohoni perspective (even where some might consider it obsolete), and only later discuss more recent developments. Also, while the questions raised by GCT have apparently sparked quite a bit of interesting progress in pure mathematics—much of it only marginally related to complexity theory—in this section I'll concentrate exclusively on the quest to prove circuit lower bounds.

I like to describe GCT as "the string theory of computer science." Like string theory, GCT has the aura of an intricate theoretical superstructure from the far future, impatiently being worked on today. Both have attracted interest partly because of "miraculous coincidences" (for string theory, these include anomaly cancellations and the prediction of gravitons; for GCT, exceptional properties of the permanent and determinant, and surprising algorithms to compute the multiplicities of irreps). Both have been described as deep, compelling, and even "the only game in town" (not surprisingly, a claim disputed by the fans of rival ideas!). And like with string theory, there are few parts of modern mathematics *not* known or believed to be relevant to GCT.

For both string theory and GCT, however, the central problem has been to say something novel and verifiable about the real-world phenomena that motivated the theory in the first place, and to do so in a way that depends essentially on the theory's core tenets (rather than on inspirations or analogies, or on small fragments of the theory). For string theory, that would mean making confirmed predictions or (e.g.) explaining the masses and generations of the elementary particles; for GCT, it would mean proving new circuit lower bounds. Indeed, proponents of both theories freely admit that one might spend one's whole career on the theory, without living to see a payoff of that kind.[75]

GCT is not an easy subject, and I don't pretend to be an expert. Part of the difficulty is inherent, while part of it is that the primary literature on GCT is not optimized for beginners: it contains a profusion of ideas, extended analogies, speculations, theorems, speculations given acronyms and then referred to *as if* they were theorems, and advanced mathematics assumed as background.

Mulmuley regards the beginning of GCT as a 1997 paper of his [175], which used algebraic geometry to prove a circuit lower bound for the classic MaxFlow problem. In MaxFlow, we're given as input a description of an n-vertex directed graph G, with nonnegative integer weights

[75] Furthermore, just as string theory didn't predict what new data there *has* been in fundamental physics in recent decades (e.g., the dark energy), so GCT played no role in, e.g., the proof of $\mathsf{NEXP} \not\subset \mathsf{ACC}$ (Section 6.4.2), or the breakthrough lower bounds for small-depth circuits computing the permanent (Section 6.5.2). In both cases, to point this out is to hold the theory to a high and possibly unfair standard, but also *ipso facto* to pay the theory a compliment.

on the edges (called *capacities*), along with designated source and sink vertices s and t, and a nonnegative integer w (the *target*). The problem is to decide whether w units of a liquid can be routed from s to t, with the amount of liquid flowing along an edge e never exceeding e's capacity. The MaxFlow problem is famously in P,[76] but the known algorithms are inherently serial, and it remains open whether they can be parallelized. More concretely, is MaxFlow in NC^1, or some other small-depth circuit class? Note that any proof of MaxFlow $\notin \mathsf{NC}^1$ would imply the spectacular separation $\mathsf{NC}^1 \neq \mathsf{P}$. Nevertheless, Mulmuley [175] managed to prove a strong lower bound for a restricted class of circuits, which captures almost all the MaxFlow algorithms known in practice.

Theorem 86 (Mulmuley [175]) *Consider an arithmetic circuit for* MaxFlow*, which takes as input the integer capacities and the integer target w.*[77] *The gates, which all have fanin 2, can perform integer addition, subtraction, and multiplication $(+, -, \times)$ as well as integer comparisons $(=, \leq, <)$. A comparison gate returns the integer 1 if it evaluates to "true" and 0 if it evaluates to "false." The circuit's final output must be 1 if the answer is "yes" and 0 if the answer is "no." Direct access to the bit-representations of the integers is not allowed, nor (for example) are the floor and ceiling functions.*

Any such circuit must have depth $\Omega(\sqrt{n})$. Moreover, this holds even if the capacities are restricted to be $O(n^2)$-bit integers.

Similar to previous arithmetic complexity lower bounds, the proof of Theorem 86 proceeds by noticing that any small-depth arithmetic circuit separates the yes-instances from the no-instances via a small number of low-degree algebraic surfaces. It then appeals to results from algebraic geometry (e.g., the Milnor-Thom Theorem) to show that, for problems of interest (such as MaxFlow), no such separation by low-degree surfaces is possible. As Mulmuley already observed in [175], the second part of the argument would work just as well for a *random* problem. Since polynomial degree is efficiently computable, this means that we get a natural proof in the Razborov-Rudich sense (Section 6.2.5). So the same technique can't possibly work to prove $\mathsf{NC}^1 \neq \mathsf{P}$: it must break down when arbitrary bit operations are allowed. Nevertheless, for Mulmuley, Theorem 86 was strong evidence that algebraic geometry provides a route to separating complexity classes.

The foundational GCT papers by Mulmuley and his collaborators, written between 2001 and 2013, are "GCT1" through "GCT8" [184, 185, 183, 48, 182, 179, 179, 176, 177], though much of the material in those papers has been rendered obsolete by subsequent developments. Readers seeking a less overwhelming introduction could try Mulmuley's overviews in *Communications of the ACM* [181] or in *Journal of the ACM* [180]; or lecture notes and other materials on Mulmuley's GCT website [174]; or surveys by Regan [211] or by Landsberg [151]. Perhaps the most beginner-friendly exposition of GCT yet written—though it covers only the early parts of the program—is contained in Joshua Grochow's PhD thesis [105, Chapter 3].

[76] MaxFlow can easily be reduced to linear programming, but Ford and Fulkerson [85] also gave a much faster and direct way to solve it, which can be found in any undergraduate algorithms textbook. There have been further improvements since.

[77] We can assume, if we like, that G, s, and t are hardwired into the circuit. We can also allow constants such as 0 and 1 as inputs, but this is not necessary, as we can also generate constants ourselves using comparison gates and arithmetic.

6.6.1 From Complexity to Algebraic Geometry

So what *is* GCT? It's easiest to understand GCT as a program to prove Valiant's Conjecture 70—that is, to show that any affine embedding of the $n \times n$ permanent over $\mathbb{C}$ into the $m \times m$ determinant over $\mathbb{C}$ requires (say) $m = 2^{n^{\Omega(1)}}$, and hence, that the permanent requires exponential-size arithmetic circuits. GCT also includes an even more speculative program to prove Boolean lower bounds, such as $\mathsf{NP} \not\subset \mathsf{P/poly}$ and hence $\mathsf{P} \neq \mathsf{NP}$. However, if we accept the premises of GCT in the first place (e.g., the primacy of algebraic geometry for circuit lower bounds), then we might as well start with the permanent versus determinant problem, since that's where the core ideas of GCT come through the most clearly.

The first observation Mulmuley and Sohoni make is that Valiant's Conjecture 70 can be translated into an algebraic-geometry conjecture about *orbit closures*. In more detail, consider a group G that acts on a set V: in the examples that interest us, G will be a group of complex matrices, while V will be a high-dimensional vector space over $\mathbb{C}$ (e.g., the space of all homogeneous degree-n polynomials over some set of variables). Then the *orbit* of a point $v \in V$, denoted Gv, is the set $\{g \cdot v : g \in G\}$. Also, the *orbit closure* of x, denoted $\overline{Gv}$, is the closure of Gv in the usual complex topology. It contains all the points that can be arbitrarily well approximated by points in Gv.

To be more concrete, let $G = \mathrm{GL}_{m^2}(\mathbb{C})$ be the general linear group: the group of all invertible $m^2 \times m^2$ complex matrices. Also, let V be the vector space of all homogeneous degree-m complex polynomials over m^2 variables; the variables are labeled $x_{1,1}, \ldots, x_{m,m}$, and we'll think of them as the entries of an $m \times m$ matrix X. (Recall that a *homogeneous* polynomial is one where all the monomials have the same degree m; restricting to them lets GCT talk about linear maps rather than affine ones.) Then G acts on V in an obvious way: for all matrices $A \in G$ and polynomials $p \in V$, we can set $(A \cdot p)(x) := p(Ax)$. Indeed, this action is a *group representation*: each $A \in G$ acts linearly on the coefficients of p, so we get a homomorphism from G to the linear transformations on V.

Now, we'd like to interpret both the $m \times m$ determinant and the $n \times n$ permanent ($n \ll m$) as points in V, in order to phrase the permanent versus determinant problem in terms of orbit closures. For the determinant, this is trivial: Det_m *is* a degree-m homogeneous polynomial over the x_{ij}'s. The $n \times n$ permanent, on the other hand, is a lower-degree polynomial over a smaller set of variables. GCT solves that problem by considering the so-called *padded permanent*,

$$\mathrm{Per}^*_{m,n}(X) := x_{m,m}^{m-n} \, \mathrm{Per}_n(X|_n),$$

where $X|_n$ denotes the top-left $n \times n$ submatrix of X, and $x_{m,m}$ is just some entry of X that's not in $X|_n$. This is a homogeneous polynomial of degree m.

Let $\chi_{\mathrm{Det},m} := \overline{G \cdot \mathrm{Det}_m}$ be the orbit closure of the $m \times m$ determinant, and let $\chi_{\mathrm{Per},m,n} := \overline{G \cdot \mathrm{Per}^*_{m,n}}$ be the orbit closure of the padded $n \times n$ permanent. I can now state the central conjecture that GCT seeks to prove.

Conjecture 87 (Mulmuley-Sohoni Conjecture) *If $m = 2^{n^{o(1)}}$, then for all sufficiently large n we have $\mathrm{Per}^*_{m,n} \notin \chi_{\mathrm{Det},m}$, or equivalently $\chi_{\mathrm{Per},m,n} \not\subset \chi_{\mathrm{Det},m}$. In other words, the padded permanent is not in the orbit closure of the determinant.*

Mulmuley and Sohoni's first observation is that a proof of Conjecture 87 (or indeed, *any* lower bound on m better than $n^{\Omega(\log n)}$) would imply a proof of Valiant's Conjecture:

Proposition 88 ([184]) *Suppose there's an affine embedding of the $n \times n$ permanent into the $m \times m$ determinant: i.e., $D(n) \leq m$, in the notation of Section 6.5.1. Then $\text{Per}_{m,n}^* \in \chi_{\text{Det},m}$.*

Let me make two remarks about Proposition 88. First, for the proposition to hold, it's crucial that we're talking about the orbit *closure*, not just the orbit. It's easy to see that $\text{Per}_{m,n}^* \notin G \cdot \text{Det}_m$—for example, because every element of $G \cdot \text{Det}_m$ is *irreducible* (can't be factored into lower-degree polynomials), whereas the padded permanent is clearly reducible. But that tells us only that there's no *invertible* linear transformation of the variables that turns the determinant into the padded permanent, not that there's no linear transformation at all. In GCT, linear changes of variable play the role of reductions—so, while the orbit of f plays the role of the "f-complete problems," it's the orbit closure that plays the role of the complexity class of functions reducible to f.[78]

The second remark is that, despite their similarity, it's unknown whether Conjecture 87 is *equivalent* to Valiant's conjecture that the permanent requires affine embeddings into the determinant of size $D(n) > 2^{n^{o(1)}}$. The reason is that there are points in the orbit closure of the determinant that aren't in its "endomorphism orbit" (that is, the set of polynomials that have not-necessarily-invertible linear embeddings into the determinant). In complexity terms, these are homogeneous degree-m polynomials that can be arbitrarily well approximated by determinants of $m \times m$ matrices of linear functions, but not represented exactly.

See Grochow [105] for further discussion of both issues.

6.6.2 Characterization by Symmetries

So far, it seems like all we've done is restated Valiant's Conjecture in a more abstract language and slightly generalized it. But now we come to the main insight of GCT, which is that the permanent and determinant are both special, highly symmetric functions, and it's plausible that we can leverage that fact to learn more about their orbit closures than we could if they were arbitrary functions. For starters, $\text{Per}(X)$ is symmetric under permuting X's rows or columns, transposing X, and multiplying the rows or columns by scalars that multiply to 1. That is, we have

$$\text{Per}(X) = \text{Per}\left(X^T\right) = \text{Per}(PXQ) = \text{Per}(AXB) \tag{2}$$

for all permutation matrices P and Q, and all diagonal matrices A and B such that $\text{Per}(A)\,\text{Per}(B) = 1$. The determinant has an even larger symmetry group: we have

$$\text{Det}(X) = \text{Det}\left(X^T\right) = \text{Det}(AXB) \tag{3}$$

for all matrices A and B such that $\text{Det}(A)\,\text{Det}(B) = 1$.

[78] More broadly, there are many confusing points about GCT whose resolutions require reminding ourselves that we're talking about orbit *closures*, not only about orbits. For example, it's often said that the plan of GCT is ultimately to show that the $n \times n$ permanent has "too little symmetry" to be embedded into the $m \times m$ determinant, unless m is much larger than n. But in that case, what about (say) a random arithmetic formula f of size n, which has *no* nontrivial symmetries, but which clearly *can* be embedded into the $(n+1) \times (n+1)$ determinant, by Theorem 73? Even though this f clearly isn't characterized by its symmetries, mustn't the embedding obstructions for f be a strict superset of the embedding obstructions for the permanent—since f's symmetries are a strict subset of the permanent's symmetries—and doesn't that give rise to a contradiction? The solution to the apparent paradox is that this argument *would* be valid if we were talking only about orbits, but it's not valid for orbit closures. With orbit closures, the set of obstructions doesn't depend in a simple way on the symmetries of the original function, so it's possible that an obstruction for the permanent would fail to be an obstruction for f.

But there's a further point: it turns out that the permanent and determinant are both *uniquely characterized* (up to a constant factor) by their symmetries, among all homogeneous polynomials of the same degree. More precisely:

Theorem 89 *Let p be any degree-m homogeneous polynomial in the entries of $X \in \mathbb{C}^{m \times m}$ that satisfies $p(X) = p(PXQ) = p(AXB)$ for all permutation matrices P, Q and diagonal A, B with $\mathrm{Per}(A)\,\mathrm{Per}(B) = 1$. Then $p(X) = \alpha\,\mathrm{Per}(X)$ for some $\alpha \in \mathbb{C}$. Likewise, let p be any degree-m homogeneous polynomial in the entries of $X \in \mathbb{C}^{m \times m}$ that satisfies $p(X) = p(AXB)$ for all A, B with $\mathrm{Det}(A)\,\mathrm{Det}(B) = 1$. Then $p(X) = \alpha\,\mathrm{Det}(X)$ for some $\alpha \in \mathbb{C}$.*[79]

Theorem 89 is fairly well-known in representation theory; the determinant case dates back to Frobenius. See Grochow [105, Propositions 3.4.3 and 3.4.5] for an elementary proof, using Gaussian elimination for the determinant and even simpler considerations for the permanent. Notice that we're not merely saying that any polynomial p with the same symmetry group as the permanent is a multiple of the permanent (and similarly for the determinant), but rather that any p whose symmetry group *contains* the permanent's is a multiple of the permanent.

In a sense, Theorem 89 is the linchpin of the GCT program. Among other things, it's GCT's answer to the question of how it could overcome the natural proofs barrier. For notice that, if we picked a degree-m homogeneous polynomial at random, it almost certainly *wouldn't* be uniquely characterized by its symmetries, as the permanent and determinant are.[80] Thus, if a proof that the permanent is hard relies on symmetry-characterization, we need not fear that the same proof would work for a random homogeneous polynomial, and thereby give us a way to break arithmetic pseudorandom functions (Section 6.5.3). While this isn't mentioned as often, Theorem 89 should also let GCT overcome the relativization and algebrization barriers, since (for example) a polynomial that was $\#\mathsf{P}^A$-complete for some oracle A, rather than $\#\mathsf{P}$-complete like the permanent was, wouldn't have the same symmetries as the permanent itself.

6.6.3 The Quest for Obstructions

Because the permanent and determinant are characterized by their symmetries, and because they satisfy another technical property called "partial stability," Mulmuley and Sohoni observe that a field called *geometric invariant theory* can be used to get a handle on their orbit closures. I won't explain the details of how this works (which involve something called Luna's Étale Slice Theorem [166]), but will just state the punchline.

Given a set $S \subseteq \mathbb{C}^N$, define $R[S]$, or the *coordinate ring of* S, to be the vector space of all complex polynomials $q : \mathbb{C}^N \to \mathbb{C}$, with two polynomials identified if they agree on all points $x \in S$. Then we'll be interested in $R_{\mathrm{Det}} := R[\chi_{\mathrm{Det},m}]$ and $R_{\mathrm{Per}} := R[\chi_{\mathrm{Per},m,n}]$: the coordinate rings of the orbit closures of the determinant and the padded permanent. In this case, $N = \binom{m^2+m-1}{m}$ is the dimension of the vector space of homogeneous degree-m polynomials over m^2 variables. So the coordinate rings are vector spaces of polynomials over N variables: truly enormous objects.

[79] Note that we don't even need to assume the symmetry $p(X) = p\left(X^T\right)$; that comes as a free byproduct. Also, it might seem like "cheating" that we use the permanent to state the symmetries that characterize the permanent, and likewise for the determinant. But we're just using the permanent and determinant as convenient ways to specify which matrices A, B we want, and could give slightly more awkward symmetry conditions that avoided them. (This is especially clear for the permanent, since if A is diagonal, then $\mathrm{Per}(A)$ is just the product of the diagonal entries.)

[80] See Grochow [105, Proposition 3.4.9] for a simple proof of this, via a dimension argument.

Next, let $q : \mathbb{C}^N \rightarrow \mathbb{C}$ be one of these "big" polynomials, whose inputs are the coefficients of a "small" polynomial p (such as the permanent or determinant). Then we can define an action of the general linear group, $G = \mathrm{GL}_{m^2}(\mathbb{C})$, on q, via $(A \cdot q)(p(x)) := q(p(Ax))$ for all $A \in G$. In other words, we take the action of G on the "small" polynomials p that we previously defined, and use it to induce an action on the "big" polynomials q. Notice that this action fixes the coordinate rings R_{Det} and R_{Per} (i.e., just shifts their points around), simply because the action of G fixes the orbit closures $\chi_{\mathrm{Det},m}$ and $\chi_{\mathrm{Per},m,n}$ themselves. As a consequence, the actions on G on R_{Det} and R_{Per} give us two representations of the group G: that is, homomorphisms that map the elements of G to linear transformations on the vector spaces R_{Det} and R_{Per} respectively. Call these representations ρ_{Det} and ρ_{Per} respectively.

Like most representations, ρ_{Det} and ρ_{Per} can be decomposed uniquely into direct sums of *isotypic components* (which are not further decomposable); each isotypic component, in turn, consists of an *irreducible representation*, or "irrep" (which can't be further decomposed) that occurs with some nonnegative integer *multiplicity*.[81] In particular, let $\rho : G \rightarrow \mathbb{C}^{k \times k}$ be any irrep of G. Then ρ occurs with some multiplicity, call it $\lambda_{\mathrm{Det}}(\rho)$, in ρ_{Det}, and with some possibly different multiplicity, call it $\lambda_{\mathrm{Per}}(\rho)$, in ρ_{Per}. We're now ready for the theorem that sets the stage for the rest of GCT.

Theorem 90 (Mulmuley-Sohoni [184]) *Suppose there exists an irrep ρ such that $\lambda_{\mathrm{Per}}(\rho) > \lambda_{\mathrm{Det}}(\rho)$. Then $\mathrm{Per}^*_{m,n} \notin \chi_{\mathrm{Det},m}$: that is, the padded permanent is not in the orbit closure of the determinant.*

Note that Theorem 90 is not an "if and only if": even if $\mathrm{Per}^*_{m,n} \notin \chi_{\mathrm{Det},m}$, there's no result saying that the reason must be representation-theoretic. In GCT2 [185], Mulmuley and Sohoni *conjecture* that the algebraic geometry of $\chi_{\mathrm{Det},m}$ is in some sense completely determined by its representation theory, but if true, that would have to be for reasons specific to $\chi_{\mathrm{Det},m}$ (or other "complexity-theoretic" orbit closures).

If $\lambda_{\mathrm{Per}}(\rho) > \lambda_{\mathrm{Det}}(\rho)$, then Mulmuley and Sohoni call ρ a *multiplicity obstruction* to embedding the permanent into the determinant. Any obstruction would be a *witness* to the permanent's hardness: in crude terms, it would prove that the $m \times m$ determinant has "the wrong kinds of symmetries" to express the padded $n \times n$ permanent, unless m is much larger than n. From this point forward, GCT—at least in Mulmuley and Sohoni's vision—is focused entirely on the hunt for a multiplicity obstruction.

A priori, one could imagine proving nonconstructively that an obstruction ρ must exist, without actually finding it. However, Mulmuley and Sohoni emphatically reject that approach. They want not merely any proof of Conjecture 87, but an "explicit" proof: that is, one that yields an algorithm that actually *finds* an obstruction ρ witnessing $\mathrm{Per}^*_{m,n} \notin \chi_{\mathrm{Det},m}$, in time polynomial in m and n. Alas, as you might have gathered, the representations ρ_{Det} and ρ_{Per} are fearsomely complicated objects—so even if we accept for argument's sake that obstructions exist, we seem a very long way from algorithms to find them in less than astronomical time.[82]

For now, therefore, Mulmuley and Sohoni argue that the best way to make progress toward Conjecture 87 is to work on *more and more efficient algorithms* to compute the multiplicities of

[81] An isotypic component might be decomposable into irreps in many ways, but one always gets the same number of irreps of the same type.

[82] In principle, ρ_{Det} and ρ_{Per} are infinite-dimensional representations, so an algorithm could search them forever for obstructions without halting. On the other hand, if we impose some upper bound on the degrees of the polynomials in the coordinate ring, we get an algorithm that takes "merely" doubly- or triply-exponential time.

irreps in complicated representations like ρ_{Det} and ρ_{Per}. The hope is that, in order to design those algorithms, we'll be forced to acquire such a deep understanding that we'll then know exactly where to look for a ρ such that $\lambda_{\text{Per}}(\rho) > \lambda_{\text{Det}}(\rho)$. So that's the program that's been pursued for the last decade; I'll have more to say later about where that program currently stands.

The central idea here—that the path to proving $\mathsf{P} \neq \mathsf{NP}$ will go through *discovering new algorithms*, rather than through ruling them out—is GCT's version of "ironic complexity theory," discussed in Section 6.4. What I've been calling "irony" in this survey, Mulmuley calls "The Flip" [178]: that is, flipping lower-bound problems into upper-bound problems, which we have a much better chance of solving.

Stepping back from the specifics of GCT, Mulmuley's view is that, before we prove (say) $\mathsf{NP} \not\subset \mathsf{P/poly}$, a natural intermediate goal is to find an algorithm A that takes a positive integer n as input, runs for $n^{O(1)}$ time (or even $\exp\left(n^{O(1)}\right)$ time), and then outputs a proof that 3SAT instances of size n have no circuits of size m, for some superpolynomial function m. Such an algorithm wouldn't immediately prove $\mathsf{NP} \not\subset \mathsf{P/poly}$, because we might still not know how to prove that A succeeded for every n. Even so, it would clearly be a titanic step forward, since we could run A and check that it *did* succeed for every n we chose, perhaps even for n's in the billions. At that point, we could say either that $\mathsf{NP} \not\subset \mathsf{P/poly}$, or else that $\mathsf{NP} \subset \mathsf{P/poly}$ only "kicks in" at such large values of n as to have few or no practical consequences. Furthermore, Mulmuley argues, we'd then be in a much better position to prove $\mathsf{NP} \not\subset \mathsf{P/poly}$ outright, since we'd "merely" have to analyze A, whose very existence would obviously encode enormous insight about the problem, and prove that it worked for all n.[83]

There has indeed been progress in finding efficient algorithms to compute the multiplicities of irreps, though the state-of-the-art is still extremely far from what GCT would need. To give an example, a *Littlewood-Richardson coefficient* is the multiplicity of a given irrep in a tensor product of two irreps of the general linear group $\text{GL}_n(\mathbb{C})$. In GCT3 [183], Mulmuley et al. observed that a result of Knutson and Tao [145] implies that one can use linear programming, not necessarily to compute Littlewood-Richardson coefficients in polynomial time, but at least to decide whether they're positive or zero.[84] Bürgisser and Ikenmeyer [64] later gave a faster polynomial-time algorithm for the same problem; theirs was purely combinatorial and based on maximum flow.

Note that, even if we only had efficient algorithms for positivity, those would still be useful for finding so-called *occurrence obstructions*: that is, irreps ρ such that $\lambda_{\text{Per}}(\rho) > 0$ even though $\lambda_{\text{Det}}(\rho) = 0$. Thus, the "best-case scenario" for GCT would be for the permanent's hardness to be witnessed not just by any obstructions, but by occurrence obstructions. As we'll see, in a recent breakthrough, this "best-case scenario" has been ruled out [120, 66].

In many cases, we don't even know yet how to represent the multiplicity of an irrep as a #P function: at best, we can represent it as a *difference* between two #P functions. In those cases, research effort in GCT has sought to give "positive formulas" for the multiplicities: in other words, to represent them as sums of exponentially many *nonnegative* terms, and thereby place their computation in #P itself. The hope is that a #P formula could be a first step toward a

[83] A very loose analogy: well before Andrew Wiles proved Fermat's Last Theorem [255], that $x^n + y^n = z^n$ has no nontrivial integer solutions for any $n \geq 3$, number theorists knew a reasonably efficient algorithm that took an exponent n as input, and that (in practice, in all the cases that were tried) proved FLT for that n. Using that algorithm, in 1993—just before Wiles announced his proof—Buhler et al. [59] proved FLT for all n up to 4 million.

[84] As pointed out in Section 2.2.6, there are other cases in complexity theory where deciding positivity is much easier than counting: for example, deciding whether a graph has at least one perfect matching (counting the *number* of perfect matchings is #P-complete).

polynomial-time algorithm to decide positivity. To illustrate, Blasiak et al. gave a #P formula for "two-row Kronecker coefficients" in GCT4 [48], while Ikenmeyer, Mulmuley, and Walter [119] did so for a different subclass of Kronecker coefficients.

6.6.4 GCT and $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$

Suppose—let's dream—that everything above worked out perfectly. That is, suppose GCT led to the discovery of explicit obstructions for embedding the padded permanent into the determinant, and thence to a proof of Valiant's Conjecture 70. How would GCT go even further, to prove $\mathsf{P} \neq \mathsf{NP}$?

The short answer is that Mulmuley and Sohoni [184] defined an NP function called E, as well as a P-complete[85] function called H, and showed them to be characterized by symmetries in only a slightly weaker sense than the permanent and determinant are. The E and H functions aren't nearly as natural as the permanent and determinant, but they suffice to show that $\mathsf{P} \neq \mathsf{NP}$ could in principle be proven by finding explicit representation-theoretic obstructions, which in this case would be representations associated with the orbit of E but not with the orbit of H. Alas, because E and H are functions over a *finite* field $\mathbb{F}_q$ rather than over $\mathbb{C}$, the relevant algebraic geometry and representation theory would all be over finite fields as well. This leads to mathematical questions even less well-understood (!) than the ones discussed earlier, providing some additional support for the intuition that proving $\mathsf{P} \neq \mathsf{NP}$ should be "even harder" than proving Valiant's Conjecture.

For illustration, let me now define Mulmuley and Sohoni's E function. Let X_0 and X_1 be two $n \times n$ matrices over the finite field $\mathbb{F}_q$. Also, given a binary string $s = s_1 \cdots s_n$, let X_s be the $n \times n$ matrix obtained by choosing the i^{th} column from X_0 if $s_i = 0$ or from X_1 if $s_i = 1$ for all $i \in \{1, \ldots, n\}$. We then set

$$F(X_0, X_1) := \prod_{s \in \{0,1\}^n} \mathrm{Det}(X_s),$$

and finally set

$$E(X_0, X_1) := 1 - F(X_0, X_1)^q$$

to obtain a function in $\{0, 1\}$. Testing whether $E(X_0, X_1) = 1$ is clearly an NP problem, since an NP witness is a single s such that $\mathrm{Det}(X_s) = 0$. Furthermore, Gurvits [110] showed that, at least over $\mathbb{Z}$, testing whether $F(X_0, X_1) = 0$ is NP-complete. Interestingly, it's not known whether E itself is NP-complete—though of course, to prove $\mathsf{P} \neq \mathsf{NP}$, it would suffice to put any NP problem outside P, not necessarily an NP-complete one.

The main result about E (or rather F) is the following:

Theorem 91 (see Grochow [105, Proposition 3.4.6]) *Let $p : \mathbb{F}_q^{2n^2} \to \mathbb{F}_q$ be any homogeneous, degree-$n2^n$ polynomial in the entries of X_0 and X_1 that's divisible by $\mathrm{Det}(X_0)\,\mathrm{Det}(X_1)$, and suppose every linear symmetry of F is also a linear symmetry of p. Then $p(X_0, X_1) = \alpha F(X_0, X_1)$ for some $\alpha \in \mathbb{F}_q$.*

The proof of Theorem 91 involves some basic algebraic geometry. Even setting aside GCT, it would be interesting to know whether the existence of a plausibly-hard NP problem that's characterized by its symmetries had direct complexity-theoretic applications.

[85] A language L is called P-*complete* if (1) $L \in \mathsf{P}$, and (2) every $L' \in \mathsf{P}$ can be reduced to L by some form of reduction weaker than arbitrary polynomial-time ones (LOGSPACE reductions are often used for this purpose).

One last remark: for some complexity classes, such as BQP, we currently lack candidate problems characterized by their symmetries, so even by the speculative standards of this section, it's unclear how GCT could be used to prove (say) $\mathsf{P} \neq \mathsf{BQP}$ or $\mathsf{NP} \not\subset \mathsf{BQP}$.

6.6.5 Reports from the Trenches

In the past few years, there's been a surprising amount of progress on resolving the truth or falsehood of some of GCT's main hypotheses, and on relating GCT to mainstream complexity theory.

Alas, the news has been sharply negative for the "simplest" way GCT could have worked: namely, by finding occurrence obstructions. In a 2016 breakthrough, Bürgisser, Ikenmeyer, and Panova [66], building on slightly earlier work by Ikenmeyer and Panova [120], showed that one can't use occurrence obstructions to separate $\chi_{\mathrm{Per},m,n}$ from $\chi_{\mathrm{Det},m}$, for m superpolynomially larger than n. That is, any irrep that occurs at least once in the padded permanent representation ρ_{Per}, also occurs at least once in the determinant representation ρ_{Det}.

Particularly noteworthy is one aspect of the proof of this result. The authors assume by contradiction that an occurrence obstruction exists, to separate $\chi_{\mathrm{Per},m,n}$ from $\chi_{\mathrm{Det},m}$ for some fixed n and m. They then show inductively that there would also be obstructions proving $\chi_{\mathrm{Per},m,n} \not\subset \chi_{\mathrm{Det},m}$ for larger and larger m—until finally, one would have a contradiction with (for example) Theorem 71, the result of Grenet [102] showing that $\chi_{\mathrm{Per},2^n-1,n} \subseteq \chi_{\mathrm{Det},2^n-1}$. This underscores one of the central worries about GCT since the beginning: namely, how is this approach sensitive to the *quantitative* nature of the permanent versus determinant problem? What is it that would break down, if one tried to use GCT to prove lower bounds so aggressive that they were actually false? In the case of occurrence obstructions, the answer turns out to be that nothing would break down.

On the positive side, Ikenmeyer, Mulmuley, and Walter [119] showed that there are superpolynomially many Kronecker coefficients that *do* vanish, thereby raising hope that occurrence obstructions might exist after all, if not for permanent versus determinant then perhaps for other problems. Notably, they proved this result by first giving a #P formula for the relevant class of Kronecker coefficients, thereby illustrating the GCT strategy of first looking for algorithms and only later looking for the obstructions themselves.

Also on the positive side, Landsberg and Ressayre [154] recently showed that Theorem 71—the embedding of $n \times n$ permanents into $(2^n-1) \times (2^n-1)$ determinants due to Grenet [102]—is *exactly optimal* among all embeddings that respect left-multiplication by permutation and diagonal matrices (thus, roughly half the symmetry group of the $n \times n$ permanent), as Grenet's embedding turns out to do.

Also on the positive side, we now know that the central property of the permanent and determinant that GCT seeks to exploit—namely, their characterization by symmetries—does indeed have complexity-theoretic applications. In particular, Mulmuley [178] observed that one can use the symmetry-characterization of the permanent to give a different, and in some ways nicer, proof of the classic result of Lipton [162] that the permanent is *self-testable*: that is, given a circuit C that's alleged to compute $\mathrm{Per}(X)$, in randomized polynomial time one can either verify that $C(X) = \mathrm{Per}(X)$ for most matrices X, or else find a counterexample where $C(X) \neq \mathrm{Per}(X)$.[86] Subsequently, Kayal [136] took Mulmuley's observation further to prove the following.

[86] This result of Lipton's provided the germ of the proof that $\mathsf{IP} = \mathsf{PSPACE}$; see Section 6.3.1.

Mulmuley's test improves over Lipton's by, for example, requiring only nonadaptive queries to C rather than adaptive ones.

Theorem 92 (Kayal [136]) *Let $n = d^2$, and suppose we're given black-box access to a degree-d polynomial $p : \mathbb{F}^n \rightarrow \mathbb{F}$, for some field $\mathbb{F}$. Suppose also that we're promised that $p(X) = \mathrm{Per}(L(X))$, where L is a $d \times d$ matrix of affine forms in the variables $X = (x_1, \ldots, x_n)$, and the mapping $X \rightarrow L(X)$ is invertible (that is, has rank n). Then there's a randomized algorithm, running in $n^{O(1)}$ time, that actually finds an L such that $p(X) = \mathrm{Per}(L(X))$. The same holds if $p(X) = \mathrm{Det}(L(X))$.*

In the same paper [136], Kayal also showed that if q is an *arbitrary* polynomial, then deciding whether there exist A and b such that $p(x) = q(Ax + b)$ is NP-hard. So what is it about the permanent and determinant that made the problem so much easier? The answer turns out to be their symmetry-characterization, along with more specific properties of the Lie algebras of the stabilizer groups of Per and Det.

For more about the algorithmic consequences of symmetry-characterization, see for example Grochow [105, Chapter 4], who also partially derandomized Kayal's algorithm and applied it to other problems such as matrix multiplication. In my view, an exciting challenge is to use the symmetry-characterization of the permanent, or perhaps of the E function from Section 6.6.4, to prove other new complexity results—*not* necessarily circuit lower bounds—that hopefully evade the relativization and algebrization barriers.

We also now know that nontrivial circuit lower bounds—albeit, not state-of-the-art ones—can indeed be proved by finding representation-theoretic obstructions, as GCT proposed. Recall from Section 6 that the *rank* of a 3-dimensional tensor $A \in \mathbb{F}^{n \times n \times n}$ is the smallest r such that A can be written as the sum of r rank-one tensors, $t_{ijk} = x_i y_j z_k$. If $\mathbb{F}$ is a continuous field like $\mathbb{C}$, then we can also define the *border rank* of A to be the smallest r such that A can be written as the *limit* of rank-r tensors. It's known that border rank can be strictly less than rank.[87] Border rank was introduced in 1980 by Bini, Lotti, and Romani [47] to study matrix multiplication algorithms: indeed, one could call that the first appearance of orbit closures in computational complexity, decades before GCT.

More concretely, the $n \times n$ *matrix multiplication tensor*, say over $\mathbb{C}$, is defined to be the 3-dimensional tensor $M_n \in \mathbb{C}^{n^2 \times n^2 \times n^2}$ whose $(i,j), (j,k), (i,k)$ entries are all 1, and whose remaining $n^6 - n^3$ entries are all 0. In 1973, Strassen [236] proved a key result connecting the rank of M_n to the complexity of matrix multiplication:

Theorem 93 (Strassen [236]) *The rank of M_n equals the minimum number of nonscalar multiplications in any arithmetic circuit that multiplies two $n \times n$ matrices.*

As an immediate corollary, the border rank of M_n lower-bounds the arithmetic circuit complexity of $n \times n$ matrix multiplication. Bini [46] showed, moreover, that the exponent ω of matrix multiplication is the same if calculated using rank or border rank—so in that sense, the two are asymptotically equal.

In a tour-de-force, in 2004 Landsberg [150] proved that the border rank of M_2 is 7, using differential geometry methods. A corollary was that any procedure to multiply two 2×2 matrices requires at least 7 nonscalar multiplications, precisely matching the upper bound discovered by Strassen [235] in 1969. (The trivial upper bound is 8 multiplications.) More recently, Hauenstein,

[87] A standard example is the $2 \times 2 \times 2$ tensor whose $(2,1,1)$, $(1,2,1)$, and $(1,1,2)$ entries are all 1, and whose 5 remaining entries are all 0. One can check that this tensor has a rank of 3 but border rank of 2.

Ikenmeyer, and Landsberg [117] reproved the "border rank is 7" result, using methods that are more explicit and closer to GCT (but that still don't yield multiplicity obstructions).

I can now state what Bürgisser and Ikenmeyer [65] used GCT to prove in 2013.

Theorem 94 (Bürgisser and Ikenmeyer [65]) *There are representation-theoretic (GCT) occurrence obstructions that witness that the border rank of M_n is at least $\frac{3}{2}n^2 - 2$.*[88]

By comparison, the state-of-the-art lower bound on the border rank of matrix multiplication—but without multiplicity or occurrence obstructions—is the following.

Theorem 95 (Landsberg and Ottaviani [153]) *The border rank of M_n is at least $2n^2 - n$.*[89]

Since both lower bounds are still quadratic, neither of them shows that matrix multiplication requires more than $\sim n^2$ time, but it's interesting to see what bound one can currently achieve when one insists on GCT obstructions, compared to when one doesn't insist on them.[90]

Meanwhile, in a 2014 paper, Grochow [106] has convincingly argued that most known circuit lower bounds (though not all of them) can be put into a "broadly GCT-like" format. In particular, the AC^0 and $\mathsf{AC}^0[p]$ lower bounds of Sections 6.2.3 and 6.2.4, the embedding lower bound of Mignon and Ressayre (Theorem 72), the lower bounds for small-depth arithmetic circuits and multilinear formulas of Section 6.5.2, and many other results can each be seen as constructing a *separating module*: that is, a "big polynomial" that takes as input the coefficients of an input polynomial, that vanishes for all polynomials in some complexity class $\mathcal{C}$, but that doesn't vanish for a polynomial q for which we're proving that $q \notin \mathcal{C}$.

It's important to understand that Grochow didn't show that the known circuit lower bounds yield *representation-theoretic* obstructions, only that they yield separating modules. Thus, he showed that these results fit into the "GCT program" under a very broad definition, but not under a narrower definition.

Interestingly, the lower bounds that *don't* fit into the separating module format—such as $\mathsf{MA}_{\mathsf{EXP}} \not\subset \mathsf{P/poly}$ (Theorem 54) and $\mathsf{NEXP} \not\subset \mathsf{ACC}$ (Theorem 64)—essentially all use diagonalization as a key ingredient. From the beginning, this was understood to be a major limitation of the GCT framework: that it seems not to capture diagonalization. A related point is that GCT, as it stands, has no way to take advantage of *uniformity*: for example, no way to prove $\mathsf{P} \neq \mathsf{NP}$, without also proving the stronger result $\mathsf{NP} \not\subset \mathsf{P/poly}$. However, given that we can prove $\mathsf{P} \neq \mathsf{EXP}$ but can't even prove $\mathsf{NEXP} \not\subset \mathsf{TC}^0$, it seems conceivable that uniformity could help in proving $\mathsf{P} \neq \mathsf{NP}$.

6.6.6 The Lessons of GCT

Expert opinion is divided about GCT's prospects. Some feel that GCT does little more than take complicated questions and make them even more complicated. Others feel it's a natural and

[88] On the other hand, Bürgisser and Ikenmeyer [65] also showed that this is essentially the best lower bound achievable using their techniques.

[89] Landsberg and Michalek [152] improved this very slightly, to $2n^2 - n + 1$.

[90] Of course, this situation raises the possibility that even if representation-theoretic obstructions *do* exist, proving their existence could be *even harder* than proving complexity class separations in some more direct way. One possible "failure mode" for GCT is that, after decades of struggle, mathematicians and computer scientists finally prove Valiant's Conjecture and $\mathsf{P} \neq \mathsf{NP}$—and then, after *further* decades of struggle, it's shown that GCT could've proven these results as well (albeit with quantitatively weaker bounds).

reasonable approach, and that the complication is an inevitable byproduct of finally grappling with the real issues. Of course, one can also "cheer GCT from the sidelines" without feeling prepared to work on it oneself, particularly given the unclear prospects for any computer-science payoff in the foreseeable future. (Mulmuley once told me he thought it would take a hundred years until GCT led to major complexity class separations, and he's the *optimist*!)

Personally, I'd call myself a qualified fan of GCT, in much the same way and for the same reasons that I'm a qualified fan of string theory. I think all complexity theorists should learn something about GCT—for one thing, because it has general lessons for the quest to prove $\mathsf{P} \neq \mathsf{NP}$, *even if* it ends up not succeeding, or evolving still further away from the original Mulmuley-Sohoni vision. This section is devoted to what I believe those lessons are.

A first lesson is that we can in principle evade the relativization, algebrization, and natural proofs barriers by using the existence of complete problems with special properties: as a beautiful example, the property of being "characterized by symmetries," which the permanent and determinant both enjoy. A second lesson is that "ironic complexity theory" has even further reach than one might have thought: one could use the existence of surprisingly fast algorithms, not merely to show that certain complexity collapses would violate hierarchy theorems, but also to help *find certificates* that problems are hard. A third lesson is that there's at least one route by which a circuit lower bound proof would need to know about huge swathes of "traditional, continuous" mathematics, as many computer scientists have long suspected (or feared!).

But none of those lessons really gets to the core of the matter. One of the most striking features of GCT is that, even as the approach stands today, it "knows" about various nontrivial problems in P, such as maximum flow and linear programming (because they're involved in deciding whether the multiplicities of irreps are nonzero). We knew, of course, that any proof of $\mathsf{P} \neq \mathsf{NP}$ would need to "know" that linear programming, matching, and so on are in P and are therefore different from 3SAT. (Indeed, that's one way to express the main difficulty of the $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ problem.) So the fact that GCT knows about all these polynomial-time algorithms seems reassuring. But what's strange is that GCT seems to know the *upper* bounds, not the lower bounds—the power of the algorithms, but not their limitations! In other words, consider a hypothetical proof of $\mathsf{P} \neq \mathsf{NP}$ using GCT. If we ignore the details, and look from a distance of a thousand miles, the proof seems to be telling us not: "You see how weak P is? You see all these problems it can't solve?" but rather, "You see how strong P is? You see all these amazing, nontrivial problems it *can* solve?" The proof would seem to building up an impressive case for the wrong side of the argument!

One response would be to point out that this is math, not a courtroom debate, and leave it at that. But perhaps one can do better. Let A be a hypothetical problem, like matching or linear programming, that's in P for a nontrivial reason, and that's also definable purely in terms of its symmetries, as the permanent and determinant are. Then we can define an orbit closure χ_A, which captures all problems reducible to A. By assumption, χ_A must be contained in χ_{P}, the orbit closure corresponding to a P-complete problem, such as Mulmuley and Sohoni's H function (see Section 6.6.4). And hence, there must *not* be any representation-theoretic obstruction to such a containment. In other words, if we were to compute the multiplicities $m_1, m_2, \ldots$ of all the irreps in the representation associated with χ_A, as well as the multiplicities $n_1, n_2, \ldots$ of the irreps associated with χ_{P}, we'd necessarily find that $m_i \leq n_i$ for all i. Furthermore, by the general philosophy of GCT, once we had our long-sought explicit formulas for these multiplicities, we might well be able to use those formulas to *prove* the above inequalities.

Now let's further conjecture—following GCT2 [185]—that the orbit closures χ_A and χ_{P} are

completely captured by their representation-theoretic data. In that case, by showing that there's no representation-theoretic obstruction to $\chi_A \subseteq \chi_{\mathsf{P}}$, we would have proved, nonconstructively, that there *exists* a polynomial-time algorithm for A! And for that reason, we shouldn't be surprised if the algorithmic techniques that are used to solve A (matching, linear programming, or whatever) have already implicitly appeared in getting to this point. Indeed, we should be worried if they didn't appear.[91]

More broadly, people sometimes stress that $\mathsf{P} \neq \mathsf{NP}$ is a "universal mathematical statement": it says there's no polynomial-time algorithm for 3SAT, no matter which area of math we use to construct such an algorithm. And thus, the argument goes, we shouldn't be shocked if nearly every area of math ends up playing some role in the proof.

It will immediately be objected that there are other "universal mathematical statements"—for example, Gödel's Incompleteness Theorem, or the unsolvability of the halting problem—that are nevertheless easy to prove, that don't require anything *close* to every area of math. But we might make the analogy that proving the unsolvability of the halting problem is like proving that the first player has a forced win in the game of Hex, which Piet Hein invented in 1942 and John Nash independently invented in 1947 [188].[92] In both cases, there's a clever gambit—diagonalization in the one case, "strategy-stealing" in the other—that lets us cut through what looks at first like a terrifyingly complex problem, and reason about it in a clean, abstract way. By contrast, proving $\mathsf{P} \neq \mathsf{NP}$ seems more like proving that White has the win in chess. Here the opening gambit fails, so (to switch metaphors) we're thrown immediately into the deep end of the pool, into considering *this* way Black could win, and *that* one, and *that* one; or into considering an immense list of possible polynomial-time algorithms for 3SAT.

Now, a natural reaction to this observation would be, not awe at the profundity of $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$, but rather despair. Since math is infinite, and since the possible "ideas for polynomial-time algorithms" are presumably unbounded, why doesn't the "universality" of $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ mean that the task of proving could go on forever? At what point can we ever say "enough! we've discovered enough polynomial-time algorithms; now we're ready to flip things around and proceed to proving $\mathsf{P} \neq \mathsf{NP}$"?

The earlier considerations about χ_A and χ_{P} suggest one possible answer to this question. Namely, we're ready to stop when we've discovered nontrivial polynomial-time algorithms, not for all problems in P, but for *all problems in* P *that are characterized by their symmetries*. For let B be a problem in P that isn't characterized by symmetries. Then the orbit closure χ_B is contained in χ_{P}, and if we could *prove* that $\chi_B \subseteq \chi_{\mathsf{P}}$, then we would've nonconstructively shown the existence of a polynomial-time algorithm for B. But our hypothetical $\mathsf{P} \neq \mathsf{NP}$ proof doesn't need to know about that. For since B isn't characterized by symmetries, the GCT arguments

[91] It would be interesting to find a function in P, more natural than the P-complete H function, that's completely characterized by its symmetries, and then try to understand explicitly why there's no representation-theoretic obstruction to that function's orbit closure being contained in χ_{P}—something we already know must be true.

[92] The game of Hex involves two players who take turns placing white and black stones respectively on a lattice of hexagons. White (which moves first) wins if it manages to connect the left and right sides of the lattice by a path of white stones, while Black wins if it connects the top and bottom by a path of black stones. (Note that one of the two must happen.)

As John Nash observed, there's a breathtakingly easy proof that White has a forced win in this game. Namely, suppose by contradiction that Black had the win. Then White could place a stone arbitrarily on its first move, *and thereafter simulate Black's winning strategy*. The key observation is that the stone White placed in its first move will never actually hurt it: if Black's stolen strategy calls for White to place a second stone there, then White can simply place a stone anywhere else instead.

aren't going to be able to prove $\chi_B \subseteq \chi_{\mathsf{P}}$ anyway.

The above would seem to motivate an investigation of which functions in P (or NC^1, etc.) can be characterized by their symmetries. If GCT can work at all, then the set of such functions, while presumably infinite, ought to be classifiable into a finite number of families. The speculation suggests itself that these families might roughly correspond to the different algorithmic techniques: Gaussian elimination, matching, linear programming, etc., and of course whatever other techniques haven't yet been discovered. As a concrete first step toward these lofty visions, it would be interesting to find some example of a function in P that's characterized by its symmetries, like the determinant is, and that's in P only because of the existence of nontrivial polynomial-time algorithms for (say) matching or linear programming.

6.6.7 The Only Way?

In recent years, Mulmuley has advanced the following argument [178, 181, 180]: even if GCT isn't literally the *only* way forward on $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$, still, the choice of GCT to go after explicit obstructions is in some sense *provably unavoidable*—and furthermore, GCT is the "simplest" approach to finding the explicit obstructions, so Occam's Razor all but forces us to try GCT first. I agree that GCT represents a natural attack plan. But I disagree with the claim that we have any theorem telling us that GCT's choices are inevitable, or "basically" inevitable. In this section, I'll explain why.

We can identify at least four major successive decisions that GCT makes:

(1) To prove $\mathsf{P} \neq \mathsf{NP}$, we should start by proving Valiant's Conjecture 70.

(2) To prove Valiant's Conjecture, the natural approach is to prove Conjecture 87, about orbit closures.

(3) To prove Conjecture 87, the natural approach is to find explicit representation-theoretic embedding obstructions.

(4) To find those obstructions, we should start by finding faster algorithms (or algorithms in lower complexity classes) to learn about the multiplicities of irreps.

All four decisions are reasonable, but not one is obvious. And of course, even if every proposition in a list had high probability individually (or high probability conditioned on its predecessors), their conjunction could have probability close to zero!

As we saw in Section 6.5, decision (1) predates GCT by decades, so there's no need to revisit it here. Meanwhile, decision (2) seems to involve only a small strengthening of what needs to be proved, in return for a large gain in elegance. But there's plenty to question about decisions (3) and (4).

Regarding (3): we saw, in Section 6.6.5, that even if a given embedding of orbit closures is impossible, the reason might simply not be reflected in representation theory—and even if it is, it might be *harder* to prove that there's a representation-theoretic obstruction than that there's some other obstruction, and one might get only a weaker lower bound that way. At least, that's what seems to be true so far with the border rank of the matrix multiplication tensor. This is an especially acute concern because of the recent work of Bürgisser, Ikenmeyer, and Panova [66], discussed earlier, which kills the hope of finding occurrence obstructions (rather than the more general multiplicity obstructions).

But let me concentrate on (4). Is it clear that we must, in Mulmuley's words, "go for explicitness": that is, look for an efficient algorithm that takes a specific n and m as input, and tries to find a witness that it's impossible to embed the padded $n \times n$ permanent into the $m \times m$ determinant? Why not just look directly for a *proof*, which (if we found it) would work for arbitrary n and $m = n^{O(1)}$?

Mulmuley's argument for explicitness rests on what he calls the "flip theorems" [178]. These theorems, in his interpretation, assert that *any* successful approach to circuit lower bounds (not just GCT) will yield explicit obstructions as a byproduct. And thus, all GCT is doing is bringing into the open what any proof of Valiant's Conjecture or $\mathsf{NP} \not\subset \mathsf{P/poly}$ will eventually need to confront anyway.

Let me now state some of the flip theorems. First, building on a 1996 learning algorithm of Bshouty et al. [58], in 2003 Fortnow, Pavan, and Sengupta showed that, if NP-complete problems are hard at all, then there must be short lists of instances that cause all small circuits to fail.

Theorem 96 (Fortnow, Pavan, and Sengupta [90]) *Suppose* $\mathsf{NP} \not\subset \mathsf{P/poly}$. *Then for every* n *and* k, *there's a list of* 3SAT *instances* $\varphi_1, \ldots, \varphi_\ell$, *of length at most* $\ell = n^{O(1)}$, *such that every circuit* C *of size at most* n^k *fails to decide at least one* φ_i *in the list. Furthermore, such a list can be found in the class* $\mathsf{BPP}^{\mathsf{NP}}$: *that is, by a probabilistic polynomial-time Turing machine with an* NP *oracle.*[93]

Atserias [32] showed that, in the statement of Theorem 96, we can also swap the NP oracle for the circuit C itself: in other words, the list $\varphi_1, \ldots, \varphi_\ell$ can also be found in BPP^C, in probabilistic polynomial time with a C-oracle.

Likewise, if the permanent is hard, then there must be short lists of matrices that cause all small arithmetic circuits to fail—and here the lists are much easier to find than they are in the Boolean case.

Theorem 97 (Mulmuley [178, 180]) *Suppose Valiant's Conjecture 70 holds (i.e., the permanent has no polynomial-size arithmetic circuits, over finite fields* $\mathbb{F}$ *with* $|\mathbb{F}| \gg n$). *Then for every* n *and* k, *there's a list of matrices* $A_1, \ldots, A_\ell \in \mathbb{F}^{n \times n}$, *of length at most* $\ell = n^{O(1)}$, *such that for every arithmetic circuit* C *of size at most* n^k, *there exists an* i *such that* $C(A_i) \neq \mathrm{Per}(A_i)$. *Indeed, a random list* $A_1, \ldots, A_\ell$ *will have that property with* $1 - o(1)$ *probability. Furthermore, if polynomial identity testing has a black-box derandomization,*[94] *then such a list can be found in deterministic* $n^{O(1)}$ *time.*

While not hard to prove, Theorems 96 and 97 are conceptually interesting: they show that the entire hardness of 3SAT and of the permanent can be "concentrated" into a small number of instances. My difficulty is that *this* sort of "explicit obstruction" to computing 3SAT or the permanent, seems barely related to the sorts of explicit obstructions that GCT is seeking. The

[93] In fact, the list can be found in the class $\mathsf{ZPP}^{\mathsf{NP}}$, where ZPP stands for Zero-Error Probabilistic Polynomial-Time. This means that, whenever the randomized algorithm succeeds in constructing the list, it's *certain* that it's done so. (Note that referring to $\mathsf{BPP}^{\mathsf{NP}}$ and $\mathsf{ZPP}^{\mathsf{NP}}$ here is strictly an abuse of notation, since these are classes of decision problems—but theoretical computer scientists will typically perpetrate such abuses when there's no risk of confusion.)

[94] The polynomial identity testing problem was defined in Section 5.4. Also, by a "black-box derandomization," we mean a deterministic polynomial-time algorithm that outputs a *hitting set*: that is, a list of points $x_1, \ldots, x_\ell$ such that, for all small arithmetic circuits C that don't compute the identically-zero polynomial, there exists an i such that $C(x_i) \neq 0$. What makes the derandomization "black-box" is that the choice of $x_1, \ldots, x_\ell$ doesn't depend on C.

obstructions of Theorems 96 and 97 aren't representation-theoretic; they're simply lists of hard instances. Furthermore, a list like $\varphi_1, \ldots, \varphi_\ell$ or $A_1, \ldots, A_\ell$ is *not* an easy-to-verify witness that 3SAT or the permanent is hard, because we'd still need to check that the list worked against all of the exponentially many $n^{O(1)}$-sized circuits. Having such a list reduces a two-quantifier (Π_2^{P}) problem to a one-quantifier (NP) problem, but it still doesn't put the problem in P—and we have no result saying that if, for example, the permanent is hard, then there must be obstructions that can be verified in $n^{O(1)}$ time. Perhaps the best we can say is that, if we *proved* the permanent was hard, then we'd immediately get, for every n, an "obstruction" that could be both found and verified in 0 time steps! But for all we know, the complexity of finding provable obstructions could jump from $\exp\left(n^{O(1)}\right)$ to 0 as our understanding improved, without ever passing through $n^{O(1)}$.

Thus, I find, GCT's suggestion to look for faster obstruction-finding (or obstruction-recognizing) algorithms is a useful *guide*, a heuristic, a way to organize our thoughts about how we're going to find, say, an irrep ρ whose multiplicity blocks the embedding of the permanent into the determinant. But this is not the only path that should be considered.

7 Conclusions

Some will say that this survey's very length, the bewildering zoo of approaches and variations and results and barriers that it covered, is a sign that no one has any real clue about the $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ problem—or at least, that *I* don't. Among those who think that, perhaps someone will write a shorter survey that points unambiguously to the right way forward!

But it's also possible that this business, of proving brute-force search unavoidable, seems complicated because it *is* complicated. I confess to limited sympathy for the idea that someone will just set aside everything that's already known, think hard about the structural properties of the sets of languages accepted by deterministic and nondeterministic polynomial-time Turing machines, and find a property that holds for one set but not the other, thereby proving $\mathsf{P} \neq \mathsf{NP}$. For I keep coming back to the question: if a hands-off, aprioristic approach sufficed for $\mathsf{P} \neq \mathsf{NP}$, then why did it apparently *not* suffice for all the weaker separations that we've surveyed here?

At the same time, I hope our tour of the progress in lower bounds has made the case that there's no reason (yet!) to elevate $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ to some plane of metaphysical unanswerability, or assume it to be independent of the axioms of set theory, or anything like that. The experience of complexity theory, including the superpolynomial lower bounds that people *did* prove after struggle and effort, is consistent with $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ being "merely a math problem"—albeit, a math problem that happens to be well beyond the current abilities of civilization, much like the solvability of the quintic in the 1500s, or Fermat's Last Theorem in the 1700s. When we're faced with such a problem, a natural response is to want to deepen our understanding of the entire subject (in this case, algorithms and computation) surrounding the problem—not merely because that's a prerequisite to someday capturing the famous beast, but because *regardless*, the new knowledge gained along the way will hopefully find uses elsewhere. In our case, modern cryptography, quantum computing, and parts of machine learning could all be seen as flowers that bloomed in the garden of $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$.

Obviously I don't know how $\mathsf{P} \neq \mathsf{NP}$ will ultimately be proved—if I did, this would be a different survey! It seems plausible that a successful approach might yield the stronger result $\mathsf{NP} \not\subset \mathsf{P/poly}$ (i.e., that it wouldn't take advantage of uniformity); that it might start by proving Valiant's Conjecture (i.e., that the algebraic case would precede the Boolean one); and that it

might draw on many areas of mathematics, but none of these things are even close to certain. Half-jokingly, I once remarked that those who still think seriously about $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ now seem to be divided into "Team Yellow Books" (those hoping to use algebraic geometry, representation theory, and other sophisticated mathematics) and "Team Caffeinated Alien Reductions" (those hoping to use elementary yet convoluted and alien chains of logic, as in Williams's proof of $\mathsf{NEXP} \not\subset \mathsf{ACC}$)—and that as far as I can tell, the two teams seem right now to be about evenly matched.

It also seems plausible that a proof of $\mathsf{P} \neq \mathsf{NP}$ would be a beginning rather than an end—i.e., that the proof would require new insights about computation that then could be applied to many other problems besides $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$. But given how far we are from the proof, it's obviously premature to speculate about the fallout.

The one prediction I feel confident in making is that the idea of "ironic complexity theory"—i.e., of a profound duality between upper and lower bounds, where the way to prove that there's no fast algorithm for problem A is to *discover* fast algorithms for problems B, C, and D—is here to stay. As we saw, ironic complexity theory is at the core of Mulmuley's view, but it's *also* at the core of Williams's proof of $\mathsf{NEXP} \not\subset \mathsf{ACC}$, which in other respects is about as far from GCT as possible. The natural proofs barrier also provides a contrapositive version of ironic complexity, showing how the *nonexistence* of efficient algorithms is often what *prevents* us from proving lower bounds. If 3SAT is ever placed outside P, it seems like a good bet that the proof will place many other problems *inside* P—or at any rate, in smaller complexity classes than was previously known.

So, if we take an optimistic attitude (optimistic about proving intractability!), then which breakthroughs should we seek next? What's on the current horizon? There are hundreds of possible answers to that question—we've already encountered some in this survey—but if I had to highlight a few:

- Prove lower bounds against nonuniform TC^0—for example, by finding a better-than-brute-force deterministic algorithm to estimate the probability that a neural network accepts a random input (cf. Theorem 67).[95]

- Prove a lower bound better than $n^{\lfloor d/2 \rfloor}$ on the rank of an explicit d-dimensional tensor, or construct one of the many other algebraic or combinatorial objects—rigid matrices, elusive functions, etc.—that are known to imply new circuit lower bounds (see Section 6).

- Advance proof complexity to the point where we could, for example, prove a superpolynomial lower bound on the number of steps needed to convert some n-input, polynomial-size Boolean circuit C into some equivalent circuit C', via moves that each swap out a size-$O(1)$ subcircuit for a different size-$O(1)$ subcircuit with identical input/output behavior.[96]

[95] In 1999, Allender [20] showed that the permanent, and various other natural #P-complete problems, can't be solved by $\mathsf{LOGTIME}$-*uniform* TC^0 circuits: in other words, constant-depth threshold circuits for which there's an $O(\log n)$-time algorithm to output the i^{th} bit of their description, for any i. Indeed, these problems can't be solved by $\mathsf{LOGTIME}$-uniform TC^0 circuits of size $f(n)$, where f is any function that can yield an exponential when iterated a constant number of times. The proof uses a hierarchy theorem; it would be interesting to know whether it relativizes.

[96] Examples are deleting two successive NOT gates, or applying de Morgan's laws. By the completeness of Boolean algebra, one can give local transformation rules that suffice to convert any n-input Boolean circuit into any equivalent circuit using at most $\exp(n)$ moves.

From talking to experts, this problem seems closely related to the problem of proving superpolynomial lower bounds for so-called *Frege proofs*, but is possibly easier.

- Prove a superlinear or superpolynomial lower bound on the number of 2-qubit quantum gates needed to implement some explicit n-qubit unitary transformation U, to within exponential precision. (Remarkably, as far as anyone knows today, one could succeed at this without needing to prove *any* new classical circuit lower bound. On the other hand, it's plausible that one would need to overcome a unitary version of the natural proofs barrier.)

- Clarify whether ACC has a natural proofs barrier, and prove ACC lower bounds for problems in EXP or below.

- Clarify whether there's an arithmetic natural proofs barrier, or something else preventing us from crossing the "chasm at depth three" (see Sections 6.5.2 and 6.5.3).

- Prove any lower bound on $D(n)$, the determinantal complexity of the $n \times n$ permanent, better than Mignon and Ressayre's $n^2/2$ (Theorem 72).

- Derandomize polynomial identity testing—or failing that, prove *some* derandomization theorem that implies a strong new circuit lower bound.

- Discover a new class of polynomial-time algorithms, especially but not only for computing multiplicities of irreps.

- Prove any interesting new lower bound by finding a representation-theoretic obstruction to an embedding of orbit closures.

- Pinpoint additional special properties of 3SAT or the permanent that could be used to evade the natural proofs barrier, besides the few that are already known, such as symmetry-characterization, self-reducibility, and the ability to simulate machines in a hierarchy theorem.

- Perhaps my favorite challenge: *find new, semantically-interesting ways to "hobble" the classes of polynomial-time algorithms and polynomial-size circuits*, besides the ways that have already been studied, such as restricted memory, restricted circuit depth, monotone gates only, arithmetic operations only, and restricted families of algorithms (such as DPLL and certain linear and semidefinite programming relaxations). Any such restriction that one discovers is effectively a new slope that one can try to ascend up the $\mathsf{P} \neq \mathsf{NP}$ mountain.

Going even further on a limb, I've long wondered whether massive computer search could give any insight into complexity lower bound questions, beyond the relatively minor ways it's been used for this purpose already (see, e.g., [260, 257]). For example, could we feasibly discover the smallest arithmetic circuits to compute the permanents of 4×4 and 5×5 and 6×6 matrices? And if we did, would examination of those circuits yield any clues about what to prove for general n? The conventional wisdom has always been "no" to both questions. For firstly, as far as anyone knows today, the computational complexity of such a search will grow not merely exponentially but *doubly* exponentially with n (assuming the optimal circuits that we're trying to find grow exponentially themselves), and examination of the constants suggests that $n = 4$ might already be out of reach. And secondly, even if we knew the optimal circuits, they'd tell us nothing about the existence or nonexistence of clever algorithms that start to win only at much larger values of n. After all, many

of the theoretically efficient algorithms that we know today only overtake the naïve algorithms for n's in the thousands or millions.[97]

These arguments have force. Even so, I think we should be open to the possibility that someday, advances in raw computer power, and especially theoretical advances that decrease the effective size of the search space, might change the calculus and open up the field of "experimental complexity theory." One way that could happen would be if breakthroughs in GCT, or some other approach, brought the quest for "explicit obstructions" (say, to the 1000×1000 permanent having size-10^{20} circuits) within the range of computer search, if still not within the range of the human mind.

Or of course, the next great advance might come from some direction that I didn't even mention here, whether because no one grasps its importance yet or simply because *I* don't. But regardless of what happens, the best fate this survey could possibly enjoy would be to contribute, in some tiny way, to making itself obsolete.

8 Acknowledgments

I thank Andy Drucker, Daniel Grier, Joshua Grochow, Christian Ikenmeyer, Adam Klivans, Pascal Koiran, JM Landsberg, Ashley Montanaro, Bruce Smith, Leslie Valiant, Avi Wigderson, Ryan Williams, and Jon Yard for helpful observations and for answering questions. I especially thank Michail Rassias for his exponential patience with my delays completing this article.

References

[1] S. Aaronson. Is P versus NP formally independent? *Bulletin of the EATCS*, (81), October 2003.

[2] S. Aaronson. Multilinear formulas and skepticism of quantum computing. In *Proc. ACM STOC*, pages 118–127, 2004. quant-ph/0311039, www.scottaaronson.com/papers/mlinsiam.pdf.

[3] S. Aaronson. NP-complete problems and physical reality. *SIGACT News*, March 2005. quant-ph/0502072.

[4] S. Aaronson. Oracles are subtle but not malicious. In *Proc. Conference on Computational Complexity*, pages 340–354, 2006. ECCC TR05-040.

[5] S. Aaronson. Arithmetic natural proofs theory is sought, 2008. www.scottaaronson.com/blog/?p=336.

[6] S. Aaronson. *Quantum Computing Since Democritus*. Cambridge University Press, 2013.

[7] S. Aaronson. The scientific case for $P \neq NP$, 2014. www.scottaaronson.com/blog/?p=1720.

[8] S. Aaronson et al. The Complexity Zoo. www.complexityzoo.com.

[97] There are even what Richard Lipton has termed "galactic algorithms" [163], which beat their asymptotically-worse competitors, but only for values of n that likely exceed the information storage capacity of the galaxy or the observable universe. The currently-fastest matrix multiplication algorithms *might* fall into this category, although the constants don't seem to be known in enough detail to say for sure.

[9] S. Aaronson, R. Impagliazzo, and D. Moshkovitz. AM with multiple Merlins. In *Proc. Conference on Computational Complexity*, pages 44–55, 2014. arXiv:1401.6848.

[10] S. Aaronson and A. Wigderson. Algebrization: a new barrier in complexity theory. *ACM Trans. on Computation Theory*, 1(1), 2009. Earlier version in Proc. ACM STOC'2008.

[11] A. Abboud, T. D. Hansen, V. Vassilevska Williams, and R. Williams. Simulating branching programs with edit distance and friends or: a polylog shaved is a lower bound made. arXiv:1511.06022, 2015.

[12] L. Adleman. Two theorems on random polynomial time. In *Proc. IEEE FOCS*, pages 75–83, 1978.

[13] L. Adleman, J. DeMarrais, and M.-D. Huang. Quantum computability. *SIAM J. Comput.*, 26(5):1524–1540, 1997.

[14] M. Agrawal. Determinant versus permanent. In *Proceedings of the International Congress of Mathematicians*, 2006.

[15] M. Agrawal, N. Kayal, and N. Saxena. PRIMES is in P. *Annals of Mathematics*, 160(2):781–793, 2004. Preprint released in 2002.

[16] M. Agrawal and V. Vinay. Arithmetic circuits: a chasm at depth four. In *Proc. IEEE FOCS*, pages 67–75, 2008.

[17] M. Ajtai. Σ_1^1-formulae on finite structures. *Annals of Pure and Applied Logic*, 24(1):1–48, 1983.

[18] M. Ajtai. A non-linear time lower bound for Boolean branching programs. *Theory of Computing*, 1(1):149–176, 2005. Earlier version in Proc. IEEE FOCS'1999, pp. 60-70.

[19] B. Alexeev, M. A. Forbes, and J. Tsimerman. Tensor rank: some lower and upper bounds. In *Proc. Conference on Computational Complexity*, pages 283–291, 2011.

[20] E. Allender. The permanent requires large uniform threshold circuits. *Chicago Journal of Theoretical Computer Science*, 7:19, 1999.

[21] E. Allender. Cracks in the defenses: scouting out approaches on circuit lower bounds. In *Computer Science in Russia*, pages 3–10, 2008.

[22] E. Allender. A status report on the P versus NP question. *Advances in Computers*, 77:117–147, 2009.

[23] E. Allender and V. Gore. On strong separations from AC^0. In *Fundamentals of Computation Theory*, pages 1–15. Springer Berlin Heidelberg, 1991.

[24] E. Allender and V. Gore. A uniform circuit lower bound for the permanent. *SIAM J. Comput.*, 23(5):1026–1049, 1994.

[25] E. Allender and M. Koucký. Amplifying lower bounds by means of self-reducibility. *J. of the ACM*, 57(3):1–36, 2010. Earlier version in Proc. IEEE Complexity'2008, pp. 31-40.

[26] N. Alon and R. B. Boppana. The monotone circuit complexity of Boolean functions. *Combinatorica*, 7(1):1–22, 1987.

[27] A. E. Andreev. On a method for obtaining more than quadratic effective lower bounds for the complexity of π-schemes. *Moscow Univ. Math. Bull.*, 42:63–66, 1987. In Russian.

[28] S. Arora and B. Barak. *Complexity Theory: A Modern Approach*. Cambridge University Press, 2009. Online draft at www.cs.princeton.edu/theory/complexity/.

[29] S. Arora, R. Impagliazzo, and U. Vazirani. Relativizing versus nonrelativizing techniques: the role of local checkability. Manuscript, 1992.

[30] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and the hardness of approximation problems. *J. of the ACM*, 45(3):501–555, 1998. Earlier version in Proc. IEEE FOCS'1992, pp. 14-23.

[31] S. Arora and S. Safra. Probabilistic checking of proofs: a new characterization of NP. *J. of the ACM*, 45(1):70–122, 1998. Earlier version in Proc. IEEE FOCS'1992, pp. 2-13.

[32] A. Atserias. Distinguishing SAT from polynomial-size circuits, through black-box queries. In *Proc. Conference on Computational Complexity*, pages 88–95, 2006.

[33] L. Babai. Graph isomorphism in quasipolynomial time. arXiv:1512.03547, 2015.

[34] L. Babai and E. M. Luks. Canonical labeling of graphs. In *Proc. ACM STOC*, pages 171–183, 1983.

[35] A. Backurs and P. Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). In *Proc. ACM STOC*, pages 51–58, 2015.

[36] T. Baker, J. Gill, and R. Solovay. Relativizations of the P=?NP question. *SIAM J. Comput.*, 4:431–442, 1975.

[37] A. Banerjee, C. Peikert, and A. Rosen. Pseudorandom functions and lattices. In *Proc. of EUROCRYPT*, pages 719–737, 2012.

[38] P. Beame and T. Pitassi. Propositional proof complexity: past, present, and future. *Current Trends in Theoretical Computer Science*, pages 42–70, 2001.

[39] P. Beame, M. E. Saks, X. Sun, and E. Vee. Time-space trade-off lower bounds for randomized computation of decision problems. *J. of the ACM*, 50(2):154–195, 2003. Earlier version in Proc. IEEE FOCS'2000, pp. 169-179.

[40] R. Beigel and J. Tarui. On ACC. *Computational Complexity*, 4:350–366, 1994. Earlier version in Proc. IEEE FOCS'1991, pp. 783-792.

[41] S. Ben-David and S. Halevi. On the independence of P versus NP. Technical Report TR714, Technion, 1992.

[42] E. Ben-Sasson and A. Wigderson. Short proofs are narrow - resolution made simple. *J. of the ACM*, 48(2):149–169, 2001. Earlier version in Proc. IEEE Complexity'1999.

[43] C. Bennett, E. Bernstein, G. Brassard, and U. Vazirani. Strengths and weaknesses of quantum computing. *SIAM J. Comput.*, 26(5):1510–1523, 1997. quant-ph/9701001.

[44] C. H. Bennett and J. Gill. Relative to a random oracle A, $P^A \neq NP^A \neq coNP^A$ with probability 1. *SIAM J. Comput.*, 10(1):96–113, 1981.

[45] E. Bernstein and U. Vazirani. Quantum complexity theory. *SIAM J. Comput.*, 26(5):1411–1473, 1997. Earlier version in Proc. ACM STOC'1993.

[46] D. Bini. Relations between exact and approximate bilinear algorithms. applications. *Calcolo*, 17(1):87–97, 1980.

[47] D. Bini, G. Lotti, and F. Romani. Approximate solutions for the bilinear form computational problem. *SIAM J. Comput.*, 9(4):692–697, 1980.

[48] J. Blasiak, K. Mulmuley, and M. Sohoni. *Geometric complexity theory IV: nonstandard quantum group for the Kronecker problem*, volume 235 of *Memoirs of the American Mathematical Society*. 2015. arXiv:cs.CC/0703110.

[49] L. Blum, F. Cucker, M. Shub, and S. Smale. *Complexity and Real Computation*. Springer-Verlag, 1997.

[50] M. Blum. A machine-independent theory of the complexity of recursive functions. *J. of the ACM*, 14(2):322–336, 1967.

[51] A. Bogdanov and L. Trevisan. Average-case complexity. *Foundations and Trends in Theoretical Computer Science*, 2(1), 2006. ECCC TR06-073.

[52] A. D. Bookatz. QMA-complete problems. *Quantum Information and Computation*, 14(5-6):361–383, 2014. arXiv:1212.6312.

[53] R. B. Boppana, J. Håstad, and S. Zachos. Does co-NP have short interactive proofs? *Inform. Proc. Lett.*, 25:127–132, 1987.

[54] A. Braunstein, M. Mézard, and R. Zecchina. Survey propagation: An algorithm for satisfiability. *Random Structures and Algorithms*, 27(2):201–226, 2005.

[55] M. Braverman. Poly-logarithmic independence fools AC^0 circuits. In *Proc. Conference on Computational Complexity*, pages 3–8, 2009. ECCC TR09-011.

[56] R. P. Brent. The parallel evaluation of general arithmetic expressions. *J. of the ACM*, 21:201–206, 1974.

[57] N. H. Bshouty, R. Cleve, and W. Eberly. Size-depth tradeoffs for algebraic formulae. *SIAM J. Comput.*, 24(4):682–705, 1995. Earlier version in Proc. IEEE FOCS'1991, pp. 334-341.

[58] N. H. Bshouty, R. Cleve, R. Gavaldà, S. Kannan, and C. Tamon. Oracles and queries that are sufficient for exact learning. *J. Comput. Sys. Sci.*, 52(3):421–433, 1996.

[59] J. Buhler, R. Crandall, R. Ernvall, and T. Metsänkylä. Irregular primes and cyclotomic invariants to four million. *Mathematics of Computation*, 61(203):151–153, 1993.

[60] H. Buhrman, L. Fortnow, and T. Thierauf. Nonrelativizing separations. In *Proc. Conference on Computational Complexity*, pages 8–12, 1998.

[61] P. Bürgisser. Completeness and reduction in algebraic complexity theory. 2000. Available at math-www.uni-paderborn.de/agpb/work/habil.ps.

[62] P. Bürgisser. Cook's versus Valiant's hypothesis. *Theoretical Comput. Sci.*, 235(1):71–88, 2000.

[63] P. Bürgisser. On defining integers and proving arithmetic circuit lower bounds. *Computational Complexity*, 18(1):81–103, 2009. Earlier version in Proc. STACS'2007, pp. 133-144.

[64] P. Bürgisser and C. Ikenmeyer. Deciding positivity of Littlewood-Richardson coefficients. *SIAM J. Discrete Math.*, 27(4):1639–1681, 2013.

[65] P. Bürgisser and C. Ikenmeyer. Explicit lower bounds via geometric complexity theory. In *Proc. ACM STOC*, pages 141–150, 2013. arXiv:1210.8368.

[66] P. Bürgisser, C. Ikenmeyer, and G. Panova. No occurrence obstructions in geometric complexity theory. arXiv:1604.06431, 2016.

[67] S. R. Buss and R. Williams. Limits on alternation trading proofs for time-space lower bounds. *Computational Complexity*, 24(3):533–600, 2015. Earlier version in Proc. IEEE Complexity'2012, pp. 181-191.

[68] J.-Y. Cai, X. Chen, and D. Li. Quadratic lower bound for permanent vs. determinant in any characteristic. 19(1):37–56, 2010. Earlier version in Proc. ACM STOC'2008, pp. 491-498.

[69] R. Chen, R. Santhanam, and S. Srinivasan. Average-case lower bounds and satisfiability algorithms for small threshold circuits. ECCC TR15-191, 2015.

[70] X. Chen and X. Deng. The circuit-input game, natural proofs, and testing circuits with data. In *Proc. Innovations in Theoretical Computer Science (ITCS)*, pages 263–270, 2015.

[71] A. Cobham. The intrinsic computational difficulty of functions. In *Proceedings of Logic, Methodology, and Philosophy of Science II*. North Holland, 1965.

[72] S. Cook. The P versus NP problem, 2000. Clay Math Institute official problem description. At www.claymath.org/sites/default/files/pvsnp.pdf.

[73] S. A. Cook. The complexity of theorem-proving procedures. In *Proc. ACM STOC*, pages 151–158, 1971.

[74] S. A. Cook. A hierarchy for nondeterministic time complexity. In *Proc. ACM STOC*, pages 187–192, 1972.

[75] D. Coppersmith. Rapid multiplication of rectangular matrices. *SIAM J. Comput.*, 11(3):467–471, 1982.

[76] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9(3):251–280, 1990. Earlier version in Proc. ACM STOC'1987.

[77] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms (2nd edition)*. MIT Press, 2001.

[78] C. Daskalakis, P. W. Goldberg, and C. H. Papadimitriou. The complexity of computing a Nash equilibrium. *Commun. of the ACM*, 52(2):89–97, 2009. Earlier version in Proc. ACM STOC'2006.

[79] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Commun. of the ACM*, 5(7):394–397, 1962.

[80] V. Deolalikar. $P \neq NP$. Archived version available at www.win.tue.nl/~gwoegi/P-versus-NP/Deolalikar.pdf, 2010.

[81] J. Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17(3):449–467, 1965.

[82] M. R. Fellows. The Robertson-Seymour theorems: a survey of applications. *Contemporary Mathematics*, 89:1–18, 1989.

[83] S. Fiorini, S. Massar, S. Pokutta, H. R. Tiwary, and R. de Wolf. Exponential lower bounds for polytopes in combinatorial optimization. *J. of the ACM*, 62(2):17, 2015. Earlier version in Proc. ACM STOC'2012, pp. 95-106.

[84] M. Forbes, M. Kumar, and R. Saptharishi. Functional lower bounds for arithmetic circuits and connections to boolean circuit complexity. ECCC TR16-045, 2016.

[85] L. R. Ford and D. R. Fulkerson. *Flows in Networks*. Princeton, 1962.

[86] L. Fortnow. Time-space tradeoffs for satisfiability. *J. Comput. Sys. Sci.*, 60(2):337–353, 2000. Earlier version in Proc. IEEE Complexity'1997, pp. 52-60.

[87] L. Fortnow. The status of the P versus NP problem. *Commun. of the ACM*, 52(9):78–86, 2009.

[88] L. Fortnow. *The Golden Ticket: P, NP, and the Search for the Impossible*. Princeton University Press, 2009.

[89] L. Fortnow and D. van Melkebeek. Time-space tradeoffs for nondeterministic computation. In *Proc. Conference on Computational Complexity*, pages 2–13, 2000.

[90] L. Fortnow, A. Pavan, and S. Sengupta. Proving SAT does not have small circuits with an application to the two queries problem. *J. Comput. Sys. Sci.*, 74(3):358–363, 2008. Earlier version in Proc. IEEE Complexity'2003, pp. 347-350.

[91] L. Fortnow and M. Sipser. Are there interactive protocols for co-NP languages? *Inform. Proc. Lett.*, 28:249–251, 1988.

[92] A. Fraenkel and D. Lichtenstein. Computing a perfect strategy for nxn chess requires time exponential in n. *Journal of Combinatorial Theory A*, 31:199–214, 1981.

[93] H. M. Friedman. Mathematically natural concrete incompleteness. At u.osu.edu/friedman.8/files/2014/01/Putnam062115pdf-15ku867.pdf, 2015.

[94] M. Furst, J. B. Saxe, and M. Sipser. Parity, circuits, and the polynomial time hierarchy. *Math. Systems Theory*, 17:13–27, 1984. Earlier version in Proc. IEEE FOCS'1981, pp. 260-270.

[95] F. Le Gall. Powers of tensors and fast matrix multiplication. pages 296–303, 2014. arXiv:1401.7714.

[96] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[97] W. Gasarch. The P=?NP poll. *SIGACT News*, 33(2):34–47, June 2002.

[98] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *J. of the ACM*, 33(4):792–807, 1986. Earlier version in Proc. IEEE FOCS'1984, pp. 464-479.

[99] O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *J. of the ACM*, 38(1):691–729, 1991.

[100] S. Goldwasser and M. Sipser. Private coins versus public coins in interactive proof systems. In *Randomness and Computation*, volume 5 of *Advances in Computing Research*. JAI Press, 1989.

[101] R. Goodstein. On the restricted ordinal theorem. *J. Symbolic Logic*, 9:33–41, 1944.

[102] B. Grenet. An upper bound for the permanent versus determinant problem. www.lirmm.fr/~grenet/publis/Gre11.pdf, 2011.

[103] D. Grigoriev and M. Karpinski. An exponential lower bound for depth 3 arithmetic circuits. In *Proc. ACM STOC*, pages 577–582, 1998.

[104] D. Grigoriev and A. A. Razborov. Exponential lower bounds for depth 3 arithmetic circuits in algebras of functions over finite fields. *Appl. Algebra Eng. Commun. Comput.*, 10(6):465–487, 2000. Earlier version in Proc. IEEE FOCS'1998, pp. 269-278.

[105] J. A. Grochow. *Symmetry and equivalence relations in classical and geometric complexity theory*. PhD thesis, 2012.

[106] J. A. Grochow. Unifying known lower bounds via Geometric Complexity Theory. *Computational Complexity*, 24(2):393–475, 2015. Earlier version in Proc. IEEE Complexity'2014, pp. 274-285.

[107] L. K. Grover. A fast quantum mechanical algorithm for database search. In *Proc. ACM STOC*, pages 212–219, 1996. quant-ph/9605043.

[108] A. Gupta, P. Kamath, N. Kayal, and R. Saptharishi. Arithmetic circuits: a chasm at depth three. In *Proc. IEEE FOCS*, pages 578–587, 2013.

[109] A. Gupta, P. Kamath, N. Kayal, and R. Saptharishi. Approaching the chasm at depth four. *J. of the ACM*, 61(6):1–16, 2014. Earlier version in Proc. IEEE Complexity'2013, pp. 65-73.

[110] L. Gurvits. On the complexity of mixed discriminants and related problems. In *Mathematical Foundations of Computer Science*, pages 447–458, 2005.

[111] A. Haken. The intractability of resolution. *Theoretical Comput. Sci.*, 39:297–308, 1985.

[112] J. Hartmanis and R. E. Stearns. On the computational complexity of algorithms. *Transactions of the American Mathematical Society*, 117:285–306, 1965.

[113] J. Håstad. *Computational Limitations for Small Depth Circuits*. MIT Press, 1987.

[114] J. Håstad. The shrinkage exponent of De Morgan formulas is 2. *SIAM J. Comput.*, 27(1):48–64, 1998. Earlier version in Proc. IEEE FOCS'1993, pp. 114-123.

[115] J. Håstad. Some optimal inapproximability results. *J. of the ACM*, 48:798–859, 2001. Earlier version in Proc. ACM STOC'1997, pp. 1-10.

[116] J. Håstad, R. Impagliazzo, L. A. Levin, and M. Luby. A pseudorandom generator from any one-way function. *SIAM J. Comput.*, 28(4):1364–1396, 1999.

[117] J. D. Hauenstein, C. Ikenmeyer, and J. M. Landsberg. Equations for lower bounds on border rank. *Experimental Mathematics*, 22(4):372–383, 2013. arXiv:1305.0779.

[118] M. Held and R. M. Karp. A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial and Applied Mathematics*, 10(1):196–210, 1962.

[119] C. Ikenmeyer, K. Mulmuley, and M. Walter. On vanishing of Kronecker coefficients. arXiv:1507.02955, 2015.

[120] C. Ikenmeyer and G. Panova. Rectangular Kronecker coefficients and plethysms in geometric complexity theory. arXiv:1512.03798, 2015.

[121] N. Immerman. Nondeterministic space is closed under complementation. *SIAM J. Comput.*, 17(5):935–938, 1988. Earlier version in Proc. IEEE Structure in Complexity Theory, 1988.

[122] N. Immerman. *Descriptive Complexity*. Springer, 1998.

[123] R. Impagliazzo, V. Kabanets, and A. Kolokolova. An axiomatic approach to algebrization. In *Proc. ACM STOC*, pages 695–704, 2009.

[124] R. Impagliazzo, V. Kabanets, and A. Wigderson. In search of an easy witness: exponential time vs. probabilistic polynomial time. *J. Comput. Sys. Sci.*, 65(4):672–694, 2002. Earlier version in Proc. IEEE Complexity'2001, pp. 2-12.

[125] R. Impagliazzo and N. Nisan. The effect of random restrictions on formula size. *Random Structures and Algorithms*, 4(2):121–134, 1993.

[126] R. Impagliazzo and A. Wigderson. P=BPP unless E has subexponential circuits: derandomizing the XOR Lemma. In *Proc. ACM STOC*, pages 220–229, 1997.

[127] H. Jahanjou, E. Miles, and E. Viola. Local reductions. In *Proc. Intl. Colloquium on Automata, Languages, and Programming (ICALP)*, pages 749–760, 2015. arXiv:1311.3171.

[128] M. Jerrum and M. Snir. Some exact complexity results for straight-line computations over semirings. *J. of the ACM*, 29(3):874–897, 1982.

[129] V. Kabanets and J.-Y. Cai. Circuit minimization problem. In *Proc. ACM STOC*, pages 73–79, 2000. TR99-045.

[130] V. Kabanets and R. Impagliazzo. Derandomizing polynomial identity testing means proving circuit lower bounds. *Computational Complexity*, 13(1-2):1–46, 2004. Earlier version in Proc. ACM STOC'2003. ECCC TR02-055.

[131] D. M. Kane and R. Williams. Super-linear gate and super-quadratic wire lower bounds for depth-two and depth-three threshold circuits. arXiv:1511.07860, 2015.

[132] R. Kannan. Circuit-size lower bounds and non-reducibility to sparse sets. *Information and Control*, 55:40–56, 1982. Earlier version in Proc. IEEE FOCS'1981, pp. 304-309.

[133] M. Karchmer and A. Wigderson. Monotone circuits for connectivity require super-logarithmic depth. *SIAM J. Comput.*, 3:255–265, 1990. Earlier version in Proc. ACM STOC'1988, pp. 539-550.

[134] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[135] R. M. Karp and R. J. Lipton. Turing machines that take advice. *Enseign. Math.*, 28:191–201, 1982. Earlier version in Proc. ACM STOC'1980, pp. 302-309.

[136] N. Kayal. Affine projections of polynomials: extended abstract. In *Proc. ACM STOC*, pages 643–662, 2012.

[137] N. Kayal, N. Limaye, C. Saha, and S. Srinivasan. An exponential lower bound for homogeneous depth four arithmetic formulas. In *Proc. IEEE FOCS*, pages 61–70, 2014.

[138] N. Kayal, C. Saha, and R. Saptharishi. A super-polynomial lower bound for regular arithmetic formulas. In *Proc. ACM STOC*, pages 146–153, 2014.

[139] N. Kayal, C. Saha, and S. Tavenas. An almost cubic lower bound for depth three arithmetic circuits. ECCC TR16-006, 2016.

[140] S. Khot. On the Unique Games Conjecture. In *Proc. Conference on Computational Complexity*, pages 99–121, 2010.

[141] V. M. Khrapchenko. A method of determining lower bounds for the complexity of π schemes. *Matematischi Zametki*, 10:83–92, 1971. In Russian.

[142] L. Kirby and J. Paris. Accessible independence results for Peano arithmetic. *Bulletin of the London Mathematical Society*, 14:285–293, 1982.

[143] A. Klivans and D. van Melkebeek. Graph nonisomorphism has subexponential size proofs unless the polynomial-time hierarchy collapses. *SIAM J. Comput.*, 31:1501–1526, 2002. Earlier version in Proc. ACM STOC'1999.

[144] D. E. Knuth and E. G. Daylight. *Algorithmic Barriers Falling: P=NP?* Lonely Scholar, 2014.

[145] A. Knutson and T. Tao. The honeycomb model of $GL_n(\mathbb{C})$ tensor products I: proof of the saturation conjecture. *J. Amer. Math. Soc.*, 12(4):1055–1090, 1999.

[146] P. Koiran. Shallow circuits with high-powered inputs. In *Proc. Innovations in Theoretical Computer Science (ITCS)*, pages 309–320, 2011.

[147] P. Koiran. Arithmetic circuits: the chasm at depth four gets wider. *Theor. Comput. Sci.*, 448:56–65, 2012.

[148] E. Kushilevitz and N. Nisan. *Communication Complexity*. Cambridge, 1997.

[149] R. E. Ladner. On the structure of polynomial time reducibility. *J. of the ACM*, 22:155–171, 1975.

[150] J. M. Landsberg. The border rank of the multiplication of two by two matrices is seven. *J. Amer. Math. Soc.*, 19(2):447–459, 2006. arXiv:math/0407224.

[151] J. M. Landsberg. Geometric complexity theory: an introduction for geometers. *Annali dell'Universita di Ferrara*, 61(1):65–117, 2015. arXiv:1305.7387.

[152] J. M. Landsberg and M. Michalek. A $2n^2 - log(n) - 1$ lower bound for the border rank of matrix multiplication. 2016. arXiv:1608.07486.

[153] J. M. Landsberg and G. Ottaviani. New lower bounds for the border rank of matrix multiplication. *Theory of Computing*, 11:285–298, 2015. arXiv:1112.6007.

[154] J. M. Landsberg and N. Ressayre. Permanent v. determinant: an exponential lower bound assuming symmetry and a potential path towards Valiant's conjecture. arXiv:1508.05788, 2015.

[155] C. Lautemann. BPP and the polynomial hierarchy. *Inform. Proc. Lett.*, 17:215–217, 1983.

[156] J. R. Lee, P. Raghavendra, and D. Steurer. Lower bounds on the size of semidefinite programming relaxations. In *Proc. ACM STOC*, pages 567–576, 2015.

[157] L. A. Levin. Universal sequential search problems. *Problems of Information Transmission*, 9(3):115–116, 1973.

[158] L. A. Levin. Laws of information conservation (non-growth) and aspects of the foundations of probability theory. *Problems of Information Transmission*, 10(3):206–210, 1974.

[159] M. Li and P. M. B. Vitányi. Average case complexity under the universal distribution equals worst-case complexity. *Inform. Proc. Lett.*, 42(3):145–149, 1992.

[160] N. Linial, Y. Mansour, and N. Nisan. Constant depth circuits, Fourier transform, and learnability. *J. of the ACM*, 40(3):607–620, 1993. Earlier version in Proc. IEEE FOCS'1989, pp. 574-579.

[161] N. Linial and N. Nisan. Approximate inclusion-exclusion. *Combinatorica*, 10(4):349–365, 1990. Earlier version in Proc. ACM STOC'1990.

[162] R. J. Lipton. New directions in testing. In *Distributed Computing and Cryptography*, pages 191–202. AMS, 1991.

[163] R. J. Lipton. Galactic algorithms, 2010. rjlipton.wordpress.com/2010/10/23/galactic-algorithms/.

[164] R. J. Lipton and K. W. Regan. Practically P=NP?, 2014. rjlipton.wordpress.com/2014/02/28/practically-pnp/.

[165] R. J. Lipton and A. Viglas. On the complexity of SAT. In *Proc. IEEE FOCS*, pages 459–464, 1999.

[166] D. Luna. Slices étales. *Mémoires de la Société Mathématique de France*, 33:81–105, 1973.

[167] C. Lund, L. Fortnow, H. Karloff, and N. Nisan. Algebraic methods for interactive proof systems. *J. of the ACM*, 39:859–868, 1992. Earlier version in Proc. IEEE FOCS'1990, pp. 2-10.

[168] E. M. McCreight and A. R. Meyer. Classes of computable functions defined by bounds on computation: preliminary report. In *Proc. ACM STOC*, pages 79–88, 1969.

[169] D. van Melkebeek. A survey of lower bounds for satisfiability and related problems. *Foundations and Trends in Theoretical Computer Science*, 2:197–303, 2007. ECCC TR07-099.

[170] T. Mignon and N. Ressayre. A quadratic bound for the determinant and permanent problem. *International Mathematics Research Notices*, (79):4241–4253, 2004.

[171] G. L. Miller. Riemann's hypothesis and tests for primality. *J. Comput. Sys. Sci.*, 13:300–317, 1976. Earlier version in Proc. ACM STOC'1975.

[172] C. Moore and S. Mertens. *The Nature of Computation.* Oxford University Press, 2011.

[173] S. Moran. Some results on relativized deterministic and nondeterministic time hierarchies. *J. Comput. Sys. Sci.*, 22(1):1–8, 1981.

[174] K. Mulmuley. GCT publications web page. ramakrishnadas.cs.uchicago.edu.

[175] K. Mulmuley. Lower bounds in a parallel model without bit operations. *SIAM J. Comput.*, 28(4):1460–1509, 1999.

[176] K. Mulmuley. Geometric complexity theory VII: nonstandard quantum group for the plethysm problem. Technical Report TR-2007-14, University of Chicago, 2007. arXiv:0709.0749.

[177] K. Mulmuley. Geometric complexity theory VIII: on canonical bases for the nonstandard quantum groups. Technical Report TR-2007-15, University of Chicago, 2007. arXiv:0709.0751.

[178] K. Mulmuley. Explicit proofs and the flip. arXiv:1009.0246, 2010.

[179] K. Mulmuley. Geometric complexity theory VI: the flip via positivity. Technical report, University of Chicago, 2011. arXiv:0704.0229.

[180] K. Mulmuley. On P vs. NP and geometric complexity theory: dedicated to Sri Ramakrishna. *J. of the ACM*, 58(2):5, 2011.

[181] K. Mulmuley. The GCT program toward the P vs. NP problem. *Commun. of the ACM*, 55(6):98–107, June 2012.

[182] K. Mulmuley. Geometric complexity theory V: equivalence between blackbox derandomization of polynomial identity testing and derandomization of Noether's Normalization Lemma. In *Proc. IEEE FOCS*, pages 629–638, 2012. Full version available at arXiv:1209.5993.

[183] K. Mulmuley, H. Narayanan, and M. Sohoni. Geometric complexity theory III: on deciding nonvanishing of a Littlewood-Richardson coefficient. *Journal of Algebraic Combinatorics*, 36(1):103–110, 2012.

[184] K. Mulmuley and M. Sohoni. Geometric complexity theory I: An approach to the P vs. NP and related problems. *SIAM J. Comput.*, 31(2):496–526, 2001.

[185] K. Mulmuley and M. Sohoni. Geometric complexity theory II: Towards explicit obstructions for embeddings among class varieties. *SIAM J. Comput.*, 38(3):1175–1206, 2008.

[186] C. D. Murray and R. R. Williams. On the (non) NP-hardness of computing circuit complexity. In *Proc. Conference on Computational Complexity*, pages 365–380, 2015.

[187] J. Naor and M. Naor. Number-theoretic constructions of efficient pseudo-random functions. *J. of the ACM*, 51(2):231–262, 2004. Earlier version in Proc. IEEE FOCS'1997.

[188] J. Nash. Some games and machines for playing them. Technical Report D-1164, Rand Corp., 1952.

[189] J. Nash. Letter to the United States National Security Agency, 1955. Available at www.nsa.gov/public_info/_files/nash_letters/nash_letters1.pdf.

[190] M. Nielsen and I. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.

[191] N. Nisan and A. Wigderson. Lower bounds on arithmetic circuits via partial derivatives. *Computational Complexity*, 6(3):217–234, 1997. Earlier version in Proc. IEEE FOCS'1995, pp. 16-25.

[192] B. Aydınlıoğlu and E. Bach. Affine relativization: unifying the algebrization and relativization barriers. 2016.

[193] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[194] M. Paterson and U. Zwick. Shrinkage of De Morgan formulae under restriction. *Random Structures and Algorithms*, 4(2):135–150, 1993.

[195] W. J. Paul, N. Pippenger, E. Szemerédi, and W. T. Trotter. On determinism versus non-determinism and related problems. In *Proc. IEEE FOCS*, pages 429–438, 1983.

[196] C. Peikert and B. Waters. Lossy trapdoor functions and their applications. *SIAM J. Comput.*, 40(6):1803–1844, 2011. Earlier version in Proc. ACM STOC'2008.

[197] G. Perelman. The entropy formula for the Ricci flow and its geometric applications. arXiv:math/0211159, 2002.

[198] V. R. Pratt. Every prime has a succinct certificate. *SIAM J. Comput.*, 4(3):214–220, 1975.

[199] M. O. Rabin. Mathematical theory of automata. In *Proc. Sympos. Appl. Math*, volume 19, pages 153–175, 1967.

[200] M. O. Rabin. Probabilistic algorithm for testing primality. *J. Number Theory*, 12(1):128–138, 1980.

[201] R. Raz. Multi-linear formulas for permanent and determinant are of super-polynomial size. *J. of the ACM*, 56(2):8, 2009. Earlier version in Proc. ACM STOC'2004, pp. 633-641. ECCC TR03-067.

[202] R. Raz. Elusive functions and lower bounds for arithmetic circuits. *Theory of Computing*, 6:135–177, 2010. Earlier version in Proc. ACM STOC'2008.

[203] R. Raz. Tensor-rank and lower bounds for arithmetic formulas. *J. of the ACM*, 60(6):40, 2013. Earlier version in Proc. ACM STOC'2010, pp. 659-666.

[204] R. Raz and A. Yehudayoff. Lower bounds and separations for constant depth multilinear circuits. *Computational Complexity*, 18(2):171–207, 2009. Earlier version in Proc. IEEE Complexity'2008, pp. 128-139.

[205] R. Raz and A. Yehudayoff. Multilinear formulas, maximal-partition discrepancy and mixed-sources extractors. *J. Comput. Sys. Sci.*, 77(1):167–190, 2011. Earlier version in Proc. IEEE FOCS'2008, pp. 273-282.

[206] A. A. Razborov. Lower bounds for the monotone complexity of some Boolean functions. *Doklady Akademii Nauk SSSR*, 281(4):798–801, 1985. English translation in *Soviet Math. Doklady* 31:354-357, 1985.

[207] A. A. Razborov. Lower bounds on monotone complexity of the logical permanent. *Mathematical Notes*, 37(6):485–493, 1985. Original Russian version in *Matematischi Zametki*.

[208] A. A. Razborov. Lower bounds for the size of circuits of bounded depth with basis $\{\&, \oplus\}$. *Mathematicheskie Zametki*, 41(4):598–607, 1987. English translation in *Math. Notes. Acad. Sci. USSR* 41(4):333–338, 1987.

[209] A. A. Razborov. Unprovability of lower bounds on circuit size in certain fragments of bounded arithmetic. *Izvestiya Math.*, 59(1):205–227, 1995.

[210] A. A. Razborov and S. Rudich. Natural proofs. *J. Comput. Sys. Sci.*, 55(1):24–35, 1997. Earlier version in Proc. ACM STOC'1994, pp. 204-213.

[211] K. W. Regan. Understanding the Mulmuley-Sohoni approach to P vs. NP. *Bulletin of the EATCS*, 78:86–99, 2002.

[212] B. Rossman, R. A. Servedio, and L.-Y. Tan. An average-case depth hierarchy theorem for Boolean circuits. In *Proc. IEEE FOCS*, pages 1030–1048, 2015. ECCC TR15-065.

[213] T. Rothvoß. The matching polytope has exponential extension complexity. In *Proc. ACM STOC*, pages 263–272, 2014.

[214] R. Santhanam. Circuit lower bounds for Merlin-Arthur classes. In *Proc. ACM STOC*, pages 275–283, 2007.

[215] R. Santhanam and R. Williams. On uniformity and circuit lower bounds. *Computational Complexity*, 23(2):177–205, 2014. Earlier version in Proc. IEEE Complexity'2013, pp. 15-23.

[216] S. Saraf. Recent progress on lower bounds for arithmetic circuits. In *Proc. Conference on Computational Complexity*, pages 155–160, 2014.

[217] W. J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *J. Comput. Sys. Sci.*, 4(2):177–192, 1970.

[218] U. Schöning. A probabilistic algorithm for k-SAT and constraint satisfaction problems. In *Proc. IEEE FOCS*, pages 410–414, 1999.

[219] U. Schöning and R. J. Pruim. *Gems of Theoretical Computer Science*. Springer, 1998.

[220] A. Shamir. IP=PSPACE. *J. of the ACM*, 39(4):869–877, 1992. Earlier version in Proc. IEEE FOCS'1990, pp. 11-15.

[221] C. Shannon. The synthesis of two-terminal switching circuits. *Bell System Technical Journal*, 28(1):59–98, 1949.

[222] J. Shoenfield. The problem of predicativity. In Y. Bar-Hillel et al., editor, *Essays on the Foundations of Mathematics*, pages 132–142. Hebrew University Magnes Press, 1961.

[223] P. W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26(5):1484–1509, 1997. Earlier version in Proc. IEEE FOCS'1994. quant-ph/9508027.

[224] A. Shpilka and A. Wigderson. Depth-3 arithmetic circuits over fields of characteristic zero. *Computational Complexity*, 10(1):1–27, 2001. Earlier version in Proc. IEEE Complexity'1999.

[225] 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.

[226] M. Sipser. A complexity theoretic approach to randomness. In *Proc. ACM STOC*, pages 330–335, 1983.

[227] M. Sipser. The history and status of the P versus NP question. In *Proc. ACM STOC*, pages 603–618, 1992.

[228] M. Sipser. *Introduction to the Theory of Computation (Second Edition)*. Course Technology, 2005.

[229] R. Smolensky. Algebraic methods in the theory of lower bounds for Boolean circuit complexity. In *Proc. ACM STOC*, pages 77–82, 1987.

[230] R. Solovay and V. Strassen. A fast Monte-Carlo test for primality. *SIAM J. Comput.*, 6(1):84–85, 1977.

[231] A. Srinivasan. On the approximability of clique and related maximization problems. *J. Comput. Sys. Sci.*, 67(3):633–651, 2003. Earlier version in Proc. ACM STOC'2000, pp. 144-152.

[232] L. J. Stockmeyer. The complexity of approximate counting. In *Proc. ACM STOC*, pages 118–126, 1983.

[233] J. A. Storer. On the complexity of chess. *J. Comput. Sys. Sci.*, 27(1):77–100, 1983.

[234] A. J. Stothers. *On the complexity of matrix multiplication.* PhD thesis, 2010.

[235] V. Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 14(13):354–356, 1969.

[236] V. Strassen. Vermeidung von divisionen. *Journal für die Reine und Angewandte Mathematik*, 264:182–202, 1973.

[237] B. A. Subbotovskaya. Realizations of linear functions by formulas using $+, \times, -$. *Doklady Akademii Nauk SSSR*, 136(3):553–555, 1961. In Russian.

[238] E. R. Swart. P = NP. Technical report, University of Guelph, 1986. Revision in 1987.

[239] R. Szelepcsényi. The method of forced enumeration for nondeterministic automata. *Acta Informatica*, 26(3):279–284, 1988.

[240] A. Tal. Shrinkage of De Morgan formulae by spectral techniques. In *Proc. IEEE FOCS*, pages 551–560, 2014. ECCC TR14-048.

[241] É. Tardos. The gap between monotone and non-monotone circuit complexity is exponential. *Combinatorica*, 8(1):141–142, 1988.

[242] S. Tavenas. Improved bounds for reduction to depth 4 and depth 3. *Inf. Comput.*, 240:2–11, 2015. Earlier version in Proc. MFCS'2013, pp. 813-824.

[243] S. Toda. PP is as hard as the polynomial-time hierarchy. *SIAM J. Comput.*, 20(5):865–877, 1991. Earlier version in Proc. IEEE FOCS'1989, pp. 514-519.

[244] B. A. Trakhtenbrot. A survey of Russian approaches to perebor (brute-force search) algorithms. *Annals of the History of Computing*, 6(4):384–400, 1984.

[245] L. G. Valiant. Graph-theoretic arguments in low-level complexity. In *Mathematical Foundations of Computer Science*, pages 162–176, 1977.

[246] L. G. Valiant. Completeness classes in algebra. In *Proc. ACM STOC*, pages 249–261, 1979.

[247] L. G. Valiant. The complexity of computing the permanent. *Theoretical Comput. Sci.*, 8(2):189–201, 1979.

[248] L. G. Valiant. Accidental algorithms. In *Proc. IEEE FOCS*, pages 509–517, 2006.

[249] L. G. Valiant, S. Skyum, S. Berkowitz, and C. Rackoff. Fast parallel computation of polynomials using few processors. *SIAM J. Comput.*, 12(4):641–644, 1983.

[250] Various authors. Deolalikar P vs NP paper (wiki page). Last modified 30 September 2011. michaelnielsen.org/polymath1/index.php?title=Deolalikar_P_vs_NP_paper.

[251] V. Vassilevska Williams. Multiplying matrices faster than Coppersmith-Winograd. In *Proc. ACM STOC*, pages 887–898, 2012.

[252] N. V. Vinodchandran. A note on the circuit complexity of PP. *Theor. Comput. Sci.*, 347:415–418, 2005. ECCC TR04-056.

[253] R. Wagner and M. Fischer. The string-to-string correction problem. *J. of the ACM*, 21:168–178, 1974. See en.wikipedia.org/wiki/Wagner-Fischer_algorithm for independent discoveries of the same algorithm.

[254] A. Wigderson. P, NP and mathematics - a computational complexity perspective. In *Proceedings of the International Congress of Mathematicians 2006 (Madrid)*, pages 665–712. EMS Publishing House, 2007. www.math.ias.edu/~avi/PUBLICATIONS/MYPAPERS/W06/w06.pdf.

[255] A. Wiles. Modular elliptic curves and Fermat's Last Theorem. *Annals of Mathematics*, 141(3):443–551, 1995.

[256] R. Williams. Better time-space lower bounds for SAT and related problems. In *Proc. Conference on Computational Complexity*, pages 40–49, 2005.

[257] R. Williams. Applying practice to theory. *ACM SIGACT News*, 39(4):37–52, 2008.

[258] R. Williams. Time-space tradeoffs for counting NP solutions modulo integers. *Computational Complexity*, 17(2):179–219, 2008. Earlier version in Proc. IEEE Complexity'2007, pp. 70-82.

[259] R. Williams. Guest column: a casual tour around a circuit complexity bound. *ACM SIGACT News*, 42(3):54–76, 2011.

[260] R. Williams. Alternation-trading proofs, linear programming, and lower bounds. *ACM Trans. on Computation Theory*, 5(2):6, 2013. Earlier version in Proc. STACS'2010, pp. 669-680.

[261] R. Williams. Improving exhaustive search implies superpolynomial lower bounds. *SIAM J. Comput.*, 42(3):1218–1244, 2013. Earlier version in Proc. ACM STOC'2010.

[262] R. Williams. Natural proofs versus derandomization. In *Proc. ACM STOC*, pages 21–30, 2013.

[263] R. Williams. Algorithms for circuits and circuits for algorithms: connecting the tractable and intractable. In *Proceedings of the International Congress of Mathematicians*, 2014.

[264] R. Williams. New algorithms and lower bounds for circuits with linear threshold gates. In *Proc. ACM STOC*, pages 194–202, 2014.

[265] R. Williams. Nonuniform ACC circuit lower bounds. *J. of the ACM*, 61(1):1–32, 2014. Earlier version in Proc. IEEE Complexity'2011.

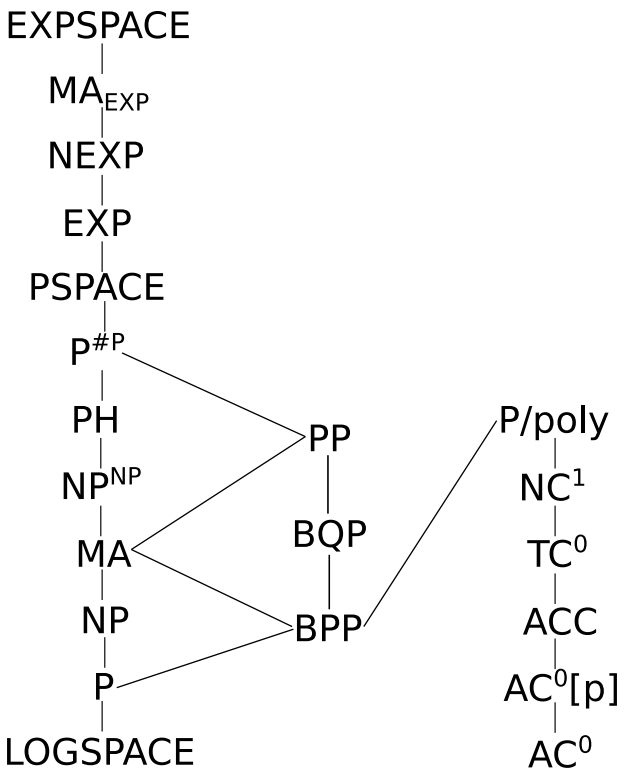

Figure 3: Known inclusion relations among 21 of the complexity classes that appear in this survey

[266] R. Williams. Strong ETH breaks with Merlin and Arthur: short non-interactive proofs of batch evaluation. ECCC TR16-002, 2016.

[267] C. B. Wilson. Relativized circuit complexity. *J. Comput. Sys. Sci.*, 31(2):169–181, 1985.

[268] M. Yannakakis. Expressing combinatorial optimization problems by linear programs. *J. Comput. Sys. Sci.*, 43(3):441–466, 1991. Earlier version in Proc. ACM STOC'1988, pp. 223-228.

[269] A. C-C. Yao. Separating the polynomial-time hierarchy by oracles (preliminary version). In *Proc. IEEE FOCS*, pages 1–10, 1985.

[270] A. C-C. Yao. On ACC and threshold circuits. In *Proc. IEEE FOCS*, pages 619–627, 1990.

9 Appendix: Glossary of Complexity Classes

To help you remember all the supporting characters in the ongoing soap opera of which P and NP are the stars, this appendix contains short definitions of the complexity classes that appear in this survey, with references to the sections where the classes are discussed in more detail. For a fuller list, containing over 500 classes, see for example my Complexity Zoo [8]. All the classes below are classes of decision problems—that is, languages $L \subseteq \{0,1\}^*$. The known inclusion relations among many of the classes are also depicted in Figure 3.

Π_2^{P}: $\mathsf{coNP}^{\mathsf{NP}}$, the second level of the polynomial hierarchy (with universal quantifier in front). See Section 2.2.3.

Σ_2^{P}: $\mathsf{NP}^{\mathsf{NP}}$, the second level of the polynomial hierarchy (with existential quantifier in front). See Section 2.2.3.

AC^0: The class decidable by a nonuniform family of polynomial-size, constant-depth, unbounded-fanin circuits of AND, OR, and NOT gates. See Section 6.2.3.

$\mathsf{AC}^0[m]$: AC^0 enhanced by MOD m gates, for some specific value of m (the case of m a prime power versus a non-prime-power are dramatically different). See Section 6.2.4.

ACC: AC^0 enhanced by MOD m gates, for every m simultaneously. See Section 6.4.2.

BPP: Bounded-Error Probabilistic Polynomial-Time. The class decidable by a polynomial-time randomized algorithm that errs with probability at most $1/3$ on each input. See Section 5.4.1.

BQP: Bounded-Error Quantum Polynomial-Time. The same as BPP except that we now allow quantum algorithms. See Section 5.5.

coNP: The class consisting of the complements of all languages in NP. Complete problems include unsatisfiability, graph non-3-colorability, etc. See Section 2.2.3.

$\mathsf{DTISP}(f(n), g(n))$: See Section 6.4.1.

EXP: Exponential-Time, or $\bigcup_k \mathsf{TIME}\left(2^{n^k}\right)$. Note the permissive definition of "exponential," which allows any polynomial in the exponent. See Section 2.2.7.

$\mathsf{EXPSPACE}$: Exponential-Space, or $\bigcup_k \mathsf{SPACE}\left(2^{n^k}\right)$. See Section 2.2.7.

IP: Interactive Proofs, the class for which a "yes" answer can be proven (to statistical certainty) via an interactive protocol in which a polynomial-time verifier Arthur exchanges a polynomial number of bits with a computationally-unbounded prover Merlin. Turns out to equal PSPACE [220]. See Section 6.3.1.

$\mathsf{LOGSPACE}$: Logarithmic-Space, or $\mathsf{SPACE}(\log n)$. Note that only read/write memory is restricted to $O(\log n)$ bits; the n-bit input itself is stored in a read-only memory. See Section 6.4.1.

MA: Merlin-Arthur, the class for which a "yes" answer can be proven to statistical certainty via a polynomial-size message from a prover ("Merlin"), which the verifier ("Arthur") then verifies in probabilistic polynomial time. Same as NP except that the verification can be probabilistic. See Section 6.3.2.

$\mathsf{MA}_{\mathsf{EXP}}$: The exponential-time analogue of MA, where now Merlin's proof can be $2^{n^{O(1)}}$ bits long, and Arthur's probabilistic verification can also take $2^{n^{O(1)}}$ time. See Section 6.3.2.

NC^1: The class decidable by a nonuniform family of polynomial-size Boolean formulas—or equivalently, polynomial-size Boolean circuits of fanin 2 and depth $O(\log n)$. The subclass of $\mathsf{P/poly}$ that is "highly parallelizable." See Section 5.2.

NEXP: Nondeterministic Exponential-Time, or $\bigcup_k \mathsf{NTIME}\left(2^{n^k}\right)$. The exponential-time analogue of NP. See Section 2.2.7.

NP: Nondeterministic Polynomial-Time, or $\bigcup_k \mathsf{NTIME}\left(n^k\right)$. The class for which a "yes" answer can be proven via a polynomial-size witness, which is verified by a deterministic polynomial-time algorithm. See Section 2.

$\mathsf{NTIME}(f(n))$: Nondeterministic $f(n)$-Time. The class for which a "yes" answer can be proven via an $O(f(n))$-bit witness, which is verified by a deterministic $O(f(n))$-time algorithm. Equivalently, the class solvable by a nondeterministic $O(f(n))$-time algorithm. See Section 2.2.7.

P: Polynomial-Time, or $\bigcup_k \mathsf{TIME}\left(n^k\right)$. The class solvable by a deterministic polynomial-time algorithm. See Section 2.

$\mathsf{P}^{\#\mathsf{P}}$: P with an oracle for $\#\mathsf{P}$ problems (i.e., for counting the exact number of accepting witnesses for any problem in NP). See Section 2.2.6.

$\mathsf{P/poly}$: P enhanced by polynomial-size "advice strings" $\{a_n\}_n$, which depend only on the input size n but can otherwise be chosen to help the algorithm as much as possible. Equivalently, the class solvable by a nonuniform family of polynomial-size Boolean circuits (i.e., a different circuit is allowed for each input size n). See Section 5.2.

PH: The Polynomial Hierarchy. The class expressible via a polynomial-time predicate with a constant number of alternating universal and existential quantifiers over polynomial-size strings. Equivalently, the union of $\Sigma_1^{\mathsf{P}} = \mathsf{NP}$, $\Pi_1^{\mathsf{P}} = \mathsf{coNP}$, $\Sigma_2^{\mathsf{P}} = \mathsf{NP}^{\mathsf{NP}}$, $\Pi_2^{\mathsf{P}} = \mathsf{coNP}^{\mathsf{NP}}$, and so on. See Section 2.2.3.

PP: Probabilistic Polynomial-Time. The class decidable by a polynomial-time randomized algorithm that, for each input x, guesses the correct answer with probability greater than $1/2$. Like BPP but without the bounded-error ($1/3$ versus $2/3$) requirement, and accordingly believed to be much more powerful. See Section 2.2.6.

PSPACE: Polynomial-Space, or $\bigcup_k \mathsf{SPACE}\left(n^k\right)$. See Section 2.2.5.

$\mathsf{SPACE}\left(f\left(n\right)\right)$: The class decidable by a serial, deterministic algorithm that uses $O\left(f\left(n\right)\right)$ bits of memory (and possibly up to $2^{O(f(n))}$ time). See Section 2.2.7.

TC^0: AC^0 enhanced by MAJORITY gates. Also corresponds to "neural networks" (polynomial-size, constant-depth circuits of threshold gates). See Section 6.2.5.

$\mathsf{TIME}\left(f\left(n\right)\right)$: The class decidable by a serial, deterministic algorithm that uses $O\left(f\left(n\right)\right)$ time steps (and therefore, $O\left(f\left(n\right)\right)$ bits of memory). See Section 2.2.7.