

\mathcal{P} , \mathcal{NP} and mathematics – a computational complexity perspective

Avi Wigderson

“ \mathcal{P} versus \mathcal{NP} – a gift to mathematics from computer science”

Steve Smale

Abstract. The \mathcal{P} versus \mathcal{NP} question distinguished itself as the central question of theoretical computer science nearly four decades ago. The quest to resolve it, and more generally, to understand the power and limits of *efficient* computation, has led to the development of computational complexity theory. While this mathematical discipline in general, and the \mathcal{P} vs. \mathcal{NP} problem in particular, have gained prominence within the mathematics community in the past decade, it is still largely viewed as a problem of computer science.

In this paper I shall try to explain why this problem, and others in computational complexity, are not only mathematical problems but also problems *about mathematics*, faced by the working mathematician. I shall describe the underlying concepts and problems, the attempts to understand and solve them, and some of the research directions this led us to. I shall explain some of the important results, as well as the major goals and conjectures which still elude us. All this will hopefully give a taste of the motivations, richness and interconnectedness of our field. I shall conclude with a few *non computational* problems, which capture \mathcal{P} vs. \mathcal{NP} and related computational complexity problems, hopefully inviting more mathematicians to attack them as well.

I believe it important to give many examples, and to underlie the intuition (and sometimes, philosophy) behind definitions and results. This may slow the pace of this article for some, in the hope to make it clearer to others.

Mathematics Subject Classification (2000). Primary 68Q15; Secondary 68Q17.

Keywords. \mathcal{P} , \mathcal{NP} , computational complexity.

1. Prelude: computation, undecidability and the limits of mathematical knowledge

Which mathematical structures can we hope to understand? Let us focus on the most basic mathematical task of classification¹. We are interested in a particular class of objects, and a particular property. We seek to *understand* which of the objects have the property and which do not. Examples include

¹This context will be general enough to be interesting and possible ramifications to other mathematical tasks are typically clear.

- (1) Which Diophantine equations have solutions?
- (2) Which knots are unknotted?
- (3) Which dynamical systems are chaotic?
- (4) Which theorems are provable in Peano arithmetic?
- (5) Which pairs of manifolds are diffeomorphic?
- (6) Which elementary statements about the Reals are true?
- (7) Which elliptic curves are modular?

A central question is what do we mean by *understanding*. When are we satisfied that our classification problem was reasonably solved? Are there problems like this which we can never solve? A central observation (popularized mainly by Hilbert) is that “satisfactory” solutions usually provide (explicitly or implicitly) procedures, which when applied to an object, determine (in finite time) if it has the property or not. Hilbert’s problems (1) and (4) above were stated, it seems, with expectation that the answer would be positive, namely that mathematicians would be able to understand them in this sense.

The breakthrough developments in the 1930s, by Gödel, Turing, Church and others led to the formal definition of an *algorithm*. This development, aside from enabling the computer revolution, made mathematically precise what Hilbert meant by a “mechanical procedure”. With it, precise theorems could be proved on the limits of our knowledge; it led to proofs that some basic mathematical problems, like (1) and (4) above will never be understood in this sense. There cannot be any *decision procedure* (an algorithm which always halts) to discern provable from unprovable statements in number theory (this was shown independently by Turing and Church), or to discern solvable from unsolvable Diophantine equations (by Davis, Putnam, Robinson and Matijasevich). These classification problems are *undecidable*.

The crucial ingredient in those (and all other undecidability) results, is showing that each of these mathematical structures can *encode computation*. This is known today to hold for many different structures in algebra, topology, geometry, analysis, logic, and more, even though a priori the structures studied seem to be completely unrelated to computation. This ubiquity makes every mathematician a potential computer scientist in disguise. We shall return to refined versions of this idea later.

Naturally, such negative results did not stop mathematical work on these structures and properties – it merely focused the necessity to understanding interesting subclasses of the given objects. Specific classes of Diophantine equations were understood much better, e.g. Fermat’s Last Theorem and the resolution of problem (7). The same holds for restricted logics for number theory, e.g. Presburger arithmetic.

The notion of a decision procedure as a minimal requirement for understanding of a mathematical problem has also led to direct positive results. It suggests that we look

for a decision procedure as *a means*, or as *first step* for understanding a problem. Thus Haken [50] showed how knots can be so understood, with his decision procedure for problem (2), and Tarski [108] showed that real-closed fields can be understood with decision procedure for problem (6). Naturally, significant *mathematical, structural* understanding was needed to develop these algorithms. Haken developed the theory of *normal surfaces*, and Tarski invented *quantifier elimination*, for their algorithms, both cornerstones of the respective fields. This only reveals the obvious: mathematical and algorithmic understanding are related and often go hand in hand. And what was true in previous centuries is truer in this one – the language of algorithms is slowly becoming competitive with the language of equations and formulas (which are special cases of algorithms) for explaining complex mathematical structures².

Now that we have seen algorithmic mathematical understanding *in principle*, it is natural to go beyond and try to quantify that level of understanding. Again, we would use a computational yardstick for it. We argue that better mathematical understanding goes hand in hand with better algorithms for “obtaining” that understanding from the given structures. To formalize it, we shall start introducing the computational terms that are central to the theory of computational complexity.

2. The computational complexity of classification (and other) problems

In this section we shall develop the basic notions of data representation, efficient computations, efficient reductions between problems, efficient verification of proofs, the classes, \mathcal{P} , \mathcal{NP} , $\text{co}\mathcal{NP}$ and \mathcal{NP} -complete problems. We shall focus on time (= number of elementary operations³ performed) as the primary resource of algorithms, when studying their efficiency. Other resources, such as memory, parallelism and more are studied in computational complexity, but we will not treat them here.

2.1. A motivating example. Let us consider the following two classification problems.

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

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

Problem (1′) is a restriction of problem (1) above. Problem (1) was undecidable, and it is natural to try to understand more restricted classes of Diophantine equations. Problem (2′) is a generalization of problem (2) above in two ways (the case of genus $g = 0$ corresponds to the knot being unknotted, and we are not restricted to knots in \mathbb{R}^3). Problem (2) was decidable, and we may want to understand (2′) even better.

²An early familiar example is Galois’ proof that roots of real polynomials of degree at least 5 have no *formula* with radicals, contrasted with Newton’s *algorithm* for approximating such roots.

³Each acting on fixed amount of data (e.g. 10 digits).

At any rate, most mathematicians would tend to agree that problems (1') and (2') have absolutely nothing to do with each other. They are from very different fields, with completely different notions, goals and tools. However, the theorem below suggests that this view may be wrong.

Theorem 2.1. *Problems (1') and (2') are equivalent.*

Moreover, the equivalence notion is natural and completely formal. Intuitively, any understanding we have of one problem, can be *simply* translated into a similar understanding of the other. The formal meaning will unfold in Subsection 2.10. To get there we need to develop the language and machinery which yield such surprising results. We start with formally defining the (finite!) representation of objects in both problems, and in general.

Consider the set of all equations of the form $Ax^2 + By + C = 0$ with integer coefficients A, B, C . A finite representation of such equation is obvious – the triple of coefficients (A, B, C) . Given such a triple – does the corresponding polynomial has a positive integer root (x, y) ? Let *2DIO* denote the subset of triples for which the answer is YES.

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

Any finite object (integers, tuples of integers, finite graphs, finite complexes, etc.)⁴ can be represented naturally by binary sequences (say over the alphabet $\{0, 1\}$). Indeed, this encoding can be done such that going back and forth between the object and its representation is simple and efficient (a notion to be formally defined below). Consequently, we let I denote the set of all finite binary sequences, and regard it as the set of inputs to all our classification problems. In this language, given a binary sequence $x \in I$ we may interpret it as a triple of integers (A, B, C) and ask if the related equation is in *2DIO*. This is problem (1'). We can also interpret x as a triple (M, K, G) of manifold, knot and integer, and ask if it is in the set *KNOT*. This is problem (2').

Theorem 2.1 states that there are *simple* translations (in both directions) between solving problem (1') and problem (2'). More precisely, it provides functions $f, h: I \rightarrow I$ performing these translations:

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

and

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

⁴A theory of algorithms which directly operate on real or complex numbers is developed in [16], which has natural parallels to some of the notions and results we shall meet.

So, if we have gained enough understanding of topology to solve e.g. the knot genus problem, it means that we automatically have gained enough number theoretic understanding for solving these quadratic Diophantine problems (and vice versa).

The translating functions f and h are called *reductions*. We capture the *simplicity* of a reduction in *computational* terms. We demand that it will be *efficiently* computable. This is what we define next.

2.2. Efficient computation and the class \mathcal{P} . In all that follows, we focus on asymptotic complexity. Thus e.g. we care neither about the time it takes to factor the number $2^{67} - 1$ (as much as Mersenne cared about it), nor about the time it takes to factor all 67-bit numbers, but rather about the asymptotic behavior of factoring n -bit numbers, as a function of the input length n . The asymptotic viewpoint is inherent to computational complexity theory, and we shall see in this article that it reveals structure which would be obscured by finite, precise analysis.

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

Definition 2.2 (The class \mathcal{P}). A function $f: I \rightarrow I$ is in the class \mathcal{P} if there is an algorithm computing f and positive constants A, c , such that for every n and every $x \in I_n$ the algorithm computes $f(x)$ in at most An^c steps.

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

This definition was suggested by Cobham [23], Edmonds [32] and Rabin [86], all attempting to formally delineate *efficient* from just finite (in their cases, exponential time) algorithms. Of course, nontrivial polynomial time algorithms were discovered earlier, long before the computer age. Many were discovered by mathematicians, who needed efficient methods to calculate (by hand). The most ancient and famous example is of course Euclid's GCD algorithm, which bypasses the factorization of the inputs when computing their common factor.

Why polynomial? The choice of polynomial time to represent efficient computation seems arbitrary, and indeed different possible choices can be made⁵. However, this particular choice has justified itself over time from many points of view. We list some important ones.

Polynomials typify “slowly growing” functions. The closure of polynomials under addition, multiplication and composition preserves the notion of efficiency under natural programming practices, such as using two programs in sequence, or using one as a subroutine of another. This choice removes the necessity to describe the

⁵and indeed were made and studied in computational complexity.

computational model precisely (e.g. it does not matter if we allow arithmetic operations only on single digits or on arbitrary integers, since long addition, subtraction, multiplication and division have simple polynomial time algorithms taught in grade school). Similarly, we need not worry about data representation: one can efficiently translate between essentially any two natural representations of a set of finite objects.

From a practical viewpoint, while a running time of, say, n^2 is far more desirable than n^{100} , very few known efficient algorithms for natural problems have exponents above 3 or 4. On the other hand, many important natural problems which so far resist efficient algorithms, cannot at present be solved faster than in *exponential* time. Thus reducing their complexity to (any) polynomial will be a huge conceptual improvement.

The importance of understanding the class \mathcal{P} is obvious. There are numerous computational problems that arise (in theory and practice) which demand efficient solutions. Many algorithmic techniques were developed in the past 4 decades and enable solving many of these problems (see e.g. the textbook [27]). These drive the ultra-fast home computer applications we now take for granted like web searching, spell checking, data processing, computer game graphics and fast arithmetic, as well as heavier duty programs used across industry, business, math and science. But many more problems yet (some of which we shall meet soon), perhaps of higher practical and theoretical value, remain elusive. The challenge of *characterizing* this fundamental mathematical object – the class \mathcal{P} of efficiently solvable problems – is far beyond us at this point.

We end this section with a few examples of nontrivial problems in \mathcal{P} of mathematical significance. In each the interplay of mathematical and computational understanding needed for the development of these algorithms is evident.

- **Primality testing.** Given an integer, determine if it is prime. Gauss literally challenged the mathematical community to find an efficient algorithm, but it took two centuries to resolve. The story of this recent achievement of [3] and its history are beautifully recounted in [46].
- **Linear programming.** Given a set of linear inequalities in many variables, determine if they are mutually consistent. This problem, and its optimization version, capture numerous others (finding optimal strategies of a zero-sum game is one) and the convex optimization techniques used to give the efficient algorithms [68], [65] for it do much more (see e.g. the books [97].)
- **Factoring polynomials.** Given a multivariate polynomial with *rational* coefficients, find its irreducible factors over \mathbb{Q} . Again, the tools developed in [73] (mainly regarding “short” bases in lattices in \mathbb{R}^n) have numerous other applications.
- **Hereditary graph properties.** Given a finite graph, test if it can be embedded on a fixed surface (like the plane or the torus). A vastly more general result is known, namely testing *any* hereditary property (one which closed under vertex

removal and edge contraction). It follows the monumental structure theory [94] of such properties, including a *finite basis theorem*. and its algorithmic versions.

- **Hyperbolic word problem.** Given any presentation of a hyperbolic group by generators and relations, and a word w in the generators, does w represent the identity element. The techniques give isoperimetric bounds on the Cayley graphs of such groups and more [47].

2.3. Efficient verification and the class \mathcal{NP} . Let $C \subset I$ be a classification problem. We are given an input $x \in I$ (describing a mathematical object) and are supposed to determine if $x \in C$ or not. It is convenient for this section to view C as defining a property; $x \in C$ are objects having the property, and $x \notin C$ are objects which do not. If we have an efficient algorithm for C , we simply apply it to x . But if we do not, what is the next best thing? One answer is, a *convincing proof* that $x \in C$. Before defining it formally, let us see a couple of motivating examples.

The first example is famous anecdote of a lecture by F. N. Cole, entitled “On the Factorization of Large Numbers”, at the 1903 AMS meeting. Without uttering a word, he went to the blackboard, wrote

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

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

What has happened there? Cole demonstrated that the number $2^{67} - 1$ is *composite*. Indeed, we can see that such a short proof can be given for any (correct) claim of the form $x \in \text{COMPOSITES}$, with *COMPOSITES* denoting the set of composite numbers. The proof is simply a nontrivial factor of x . The features we want to extract from this episode are two: The proofs are *short* and *easily verifiable*. The fact that it was extremely hard for Cole to find these factors (he said it took him “three years of Sundays”) did not affect in any way that demonstration.

A second example, which we meet daily, is what happens when we read a typical math journal paper. In it, we typically find a (claimed) theorem, followed by an (alleged) proof. Thus, we are verifying claims of the type $x \in \text{THEOREMS}$, where *THEOREMS* is the set of all provable statements in, say, set theory. It is taken for granted that the written proof is *short* (page limit) and *easily verifiable* (otherwise the referee/editor would demand clarifications), regardless how long it took to discover.

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

Definition 2.3 (The class \mathcal{NP}). The set C is in the class \mathcal{NP} if there is a function $V_C \in \mathcal{P}$ and a constant k such that

- If $x \in C$ then $\exists y$ with $|y| \leq |x|^k$ and $V_C(x, y) = 1$.
- If $x \notin C$ then $\forall y$ we have $V_C(x, y) = 0$.

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

A sequence y which “convinces” V_C that $x \in C$ is often called a *witness* or *certificate* for the membership of x in C . Again, we stress that the definition of \mathcal{NP} is not concerned with how difficult it is to come up with a witness y . Indeed, the acronym \mathcal{NP} stands for “nondeterministic polynomial time”, where the nondeterminism captures the ability of a *hypothetical* “nondeterministic” machine to “guess” a witness y (if one exists), and then verify it deterministically.

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

While it is usually the search problems which occupy us, from a computational standpoint it is often more convenient to study the decision versions. Almost always both versions are equivalent⁶.

These definitions of \mathcal{NP} were first given (independently and in slightly different forms) by Cook [24] and Levin [74]. There is much more to these seminal papers than this definition, and we shall discuss it later at length.

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

Corollary 2.4. $\mathcal{P} \subseteq \mathcal{NP}$.

A final comment is that problems in \mathcal{NP} have trivial *exponential time* algorithms. Such algorithms search through all possible short witnesses, and try to verify each. Can we always speed up this brute-force algorithm?

2.4. The \mathcal{P} vs. \mathcal{NP} question, its meaning and importance. The class \mathcal{NP} is extremely rich (we shall see examples a little later). There are literally thousands of \mathcal{NP} problems in mathematics, optimization, artificial intelligence, biology, physics, economics, industry and more which arise naturally out of different necessities, and whose efficient solutions will benefit us in numerous ways. They beg for efficient algorithms, but decades (and sometimes longer) of effort has only succeeded for a few.

⁶A notable possible exception is the set *COMPOSITES* and the suggested verification procedure to it, accepting as witness a nontrivial factor. Note that while *COMPOSITES* $\in \mathcal{P}$ as a decision problem, the related search problem is equivalent to Integer Factorization, which is not known to have an efficient algorithm.

Is it possible that *all* sets in \mathcal{NP} possess efficient algorithms, and these simply were not discovered yet? This is the celebrated \mathcal{P} vs. \mathcal{NP} question. It appeared explicitly first in the aforementioned papers of Cook and Levin, but had some informal precursors. Of particular interest is a remarkable letter written by Gödel to von Neumann about 15 years earlier which raises this fundamental question quite clearly, and shows how aware Gödel was of its significance (see the surveys [102], [51] for the original letter and translation, as well as much more on the subject at hand).

Open Problem 2.5. Is $\mathcal{P} = \mathcal{NP}$?

What explains the abundance of so many natural, important problems in the class \mathcal{NP} ? Probing the intuitive meaning of the definition of \mathcal{NP} , we see that it captures many tasks of human endeavor *for which a successful completion can be easily recognized*. Consider the following professions, and the typical tasks they are facing (this will be extremely superficial, but nevertheless instructive):

- **Mathematician:** Given a mathematical claim, come up with a proof for it.
- **Scientist:** Given a collection of data on some phenomena, find a theory explaining it.
- **Engineer:** Given a set of constraints (on cost, physical laws, etc.) come up with a design (of an engine, bridge, laptop ...) which meets these constraints.
- **Detective:** Given the crime scene, find “who’s done it”.

What is common to all this multitude of tasks is that we can typically tell a good solution when we see one (or we at least think we can). In various cases “we” may be the academic community, the customers, or the jury, but we expect the solution to be *short*, and *efficiently verifiable*, just as in the definition of \mathcal{NP} .

The richness of \mathcal{NP} follows from the simple fact that such tasks abound, and their mathematical formulation is indeed an \mathcal{NP} -problem. For all these tasks, efficiency is paramount, and so the importance of the \mathcal{P} vs. \mathcal{NP} problem is evident. The colossal implications of the possibility that $\mathcal{P} = \mathcal{NP}$ are evident as well – every instance of these tasks can be solved, optimally and efficiently.

One (psychological) reason people feel that $\mathcal{P} = \mathcal{NP}$ is unlikely, is that tasks as above often require a degree of *creativity* which we do not expect a simple computer program to have. We admire Wiles’ proof of Fermat’s Last Theorem, the scientific theories of Newton, Einstein, Darwin, Watson and Crick, the design of the Golden Gate bridge and the Pyramids, and sometimes even Hercule Poirot’s and Miss Marple’s analysis of a murder, precisely because they seem to require a leap which cannot be made by everyone, let alone a by simple mechanical device. My own view is that when we finally understand the algorithmic processes of the brain, we may indeed be able to automate the discovery of these specific achievements, and perhaps many others. But can we automate them *all*? Is it possible that *every* task for which verification

is easy, finding a solution is not much harder? If $\mathcal{P} = \mathcal{NP}$, the answer is positive, and creativity (of this abundant, verifiable kind) can be completely automated. Most computer scientists believe that this is not the case.

Conjecture 2.6. $\mathcal{P} \neq \mathcal{NP}$.

Back to mathematics! Given the discussion above, one may wonder why it is so hard to prove that indeed $\mathcal{P} \neq \mathcal{NP}$ – it seems completely obvious. We shall discuss attempts and difficulties soon, developing a methodology which will enable us to identify the *hardest* problems in \mathcal{NP} . But before that, we turn to discuss a related question with a strong relation to mathematics: the \mathcal{NP} versus $\text{co}\mathcal{NP}$ question.

2.5. The \mathcal{NP} versus $\text{co}\mathcal{NP}$ question, its meaning and importance. Fix a property $C \subseteq I$. We already have the interpretations

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

to which we now add

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

where we formally define

Definition 2.7 (The class $\text{co}\mathcal{NP}$). A set C is in the class $\text{co}\mathcal{NP}$ iff its complement $\bar{C} = I \setminus C$ is in \mathcal{NP} .

While the definition of the class \mathcal{P} is symmetric⁷, the definition of the class \mathcal{NP} is asymmetric. Having nice certificates that a given object has property C , does not automatically entail having nice certificates that a given object does not have it.

Indeed, when we can do both, we are achieving a mathematics' holy grail of understanding structure, namely *necessary and sufficient* conditions, sometimes phrased as a *duality theorem*. As we know well, such results are rare. When we insist (as we shall do) that the given certificates are *short, efficiently verifiable* ones, they are even rarer. This leads to the conjecture

Conjecture 2.8. $\mathcal{NP} \neq \text{co}\mathcal{NP}$.

First note that this conjecture implies $\mathcal{P} \neq \mathcal{NP}$. We shall discuss at length refinements of this conjecture in Section 4 on proof complexity.

Despite the shortage of such efficient complete characterizations, namely properties which are simultaneously in $\mathcal{NP} \cap \text{co}\mathcal{NP}$, they nontrivially exist. Here is a list of some exemplary ones.

⁷Having a fast algorithm to determine if an object has a property C is equivalent to having a fast algorithm for the complementary set \bar{C} .

- **Linear programming.** Systems of consistent linear inequalities.⁸
- **Zero-sum games**⁹. Finite zero-sum games in which one player can gain at least (some given value) v .
- **Graph connectivity.** The set of graphs in which *every* pair of vertices is connected by (a given number) k disjoint paths.
- **Partial order width.** Finite partial orders whose largest anti-chain has at most (a given number) w elements.
- **Primes.** Prime numbers.

These examples of problems in $\mathcal{NP} \cap \text{co}\mathcal{NP}$ were chosen to make a point. At the time of their discovery (by Farkas, von Neumann, Menger, Dilworth, and Pratt respectively) these mathematicians were seemingly interested only in characterizing these structures. It is not known if they attempted to find efficient algorithms for these problems. However all of these problems turned out to be in \mathcal{P} , with some solutions entering the pantheon of efficient algorithms (e.g. the Ellipsoid method of Khachian [68] and the Interior-Point method of Karmarkar [65], both for Linear Programming, and the recent breakthrough of Agrawal, Kayal and Saxena [3] for Primes¹⁰).

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

Can we expect this magic to always happen? Is $\mathcal{NP} \cap \text{co}\mathcal{NP} = \mathcal{P}$? At the end of Subsection 2.11 we shall see another list of problems in $\mathcal{NP} \cap \text{co}\mathcal{NP}$ which have resisted efficient algorithms for decades, and for some (e.g. factoring integers), humanity literally banks on their difficulty for electronic commerce security. Indeed, the following is generally believed:

Conjecture 2.9. $\mathcal{NP} \cap \text{co}\mathcal{NP} \neq \mathcal{P}$.

Note again that this conjecture 2.9 implies $\mathcal{P} \neq \mathcal{NP}$, but that it is independent of conjecture 2.8.

We now return to develop the main mechanism which will help us study such questions: efficient reductions.

⁸Indeed this generalizes to other convex bodies given by more general constraints, like *semi-definite* programming.

⁹This problem was later discovered to be equivalent to linear programming.

¹⁰It is interesting that assuming the Extended Riemann Hypothesis, a simple polynomial time algorithm was given 30 years earlier by Miller [78].

2.6. Reductions – a partial order of computational difficulty. In this subsection we deal with relating the computational difficulty of problems for which we have no efficient solutions (yet).

Recall that we can regard any classification problem (on finitely described objects) as a subset of our set of inputs I . Efficient reductions provide a natural partial order on such problems, that capture their relative difficulty.

Definition 2.10 (Efficient reductions). Let $C, D \subset I$ be two classification problems. $f: I \rightarrow I$ is an efficient reduction from C to D if $f \in \mathcal{P}$ and for every $x \in I$ we have $x \in C$ iff $f(x) \in D$. In this case we call f an *efficient reduction* from C to D . We write $C \leq D$ if there is an efficient reduction from C to D .

The definition of efficient computation allows two immediate observations on the usefulness of efficient reductions. First, that indeed \leq is transitive, and thus defines a partial order. Second, that if $C \leq D$ and $D \in \mathcal{P}$ then also $C \in \mathcal{P}$.

Formally, $C \leq D$ means that solving the classification problem C is *computationally* not much harder than solving D . In some cases one can replace *computationally* by the (vague) term *mathematically*. Often such usefulness in mathematical understanding requires more properties of the reduction f than merely being efficiently computable (e.g. we may want it to be represented as a linear transformation, or a low dimension polynomial map), and indeed in some cases this is possible. When such a connection between two classification problems (which look unrelated) can be proved, it can mean the importability of techniques from one area to another.

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

2.7. Completeness. We now return to classification problems. The partial order of their difficulty, provided by efficient reductions, allows us to define the *hardest* problems in a given class. Let \mathcal{C} be any collection of classification problems (namely every element of \mathcal{C} is a subset of I). Of course, here we shall mainly care about the class $\mathcal{C} = \mathcal{NP}$.

Definition 2.11 (Hardness and completeness). A problem D is called *\mathcal{C} -hard* if for every $C \in \mathcal{C}$ we have $C \leq D$. If we further have that $D \in \mathcal{C}$ then D is called *\mathcal{C} -complete*.

In other words, if D is \mathcal{C} -complete, it is a hardest problem in the class \mathcal{C} : if we manage to solve D efficiently, we have done so for all other problems in \mathcal{C} . It is not

apriori clear that a given class has any complete problems! On the other hand, a given class may have many complete problems, and by definition, they all have essentially the same complexity. If we manage to prove that *any* of them cannot be efficiently solved, then we automatically have done so for *all* of them.

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

2.8. \mathcal{NP} -completeness. As mentioned earlier, the seminal papers of Cook [24] and Levin [74] defined \mathcal{NP} , efficient reducibilities and completeness, but the crown of their achievement was the discovery of a *natural* \mathcal{NP} -complete problem.

Definition 2.12 (The problem *SAT*). A Boolean formula is a logical expression over Boolean variables (that can take values in $\{0, 1\}$) with connectives \wedge, \vee, \neg , e.g. $(x_1 \vee x_2) \wedge (\neg x_3)$. Let *SAT* denote the set of all satisfiable Boolean formulae (namely those formulae for which there is a Boolean assignment to the variables which gives it the value 1).

Theorem 2.13 ([24], [74]). *SAT is \mathcal{NP} -complete.*

We recall again the meaning of that statement. For *every* set $C \in \mathcal{NP}$ there is an efficient reduction $f: I \rightarrow I$ such that $x \in C$ iff the formula $f(x)$ is satisfiable! Furthermore, the proof gives an extra bonus which turns out to be extremely useful: given any witness y that $x \in C$ (via some verifier V_C), the same reduction converts the witness y to a Boolean assignment satisfying the formula $f(x)$. In other words, this reduction translates not only between the decision problems, but also between the associated search problems.

You might (justly) wonder how can one prove a theorem like that. Certainly the proof cannot afford to look at all problems $C \in \mathcal{NP}$ separately. The gist of the proof is a generic transformation, taking a description of the verifier V_C for C , and emulating its computation on input x and hypothetical witness y to create a Boolean formula $f(x)$ (whose variables are the bits of y). This formula simply tests the validity of the computation of V_C on (x, y) , and that this computation outputs 1. Here the locality of algorithms (say described as Turing machines) plays a central role, as checking the consistency of each step of the computation of V_C amounts simply to a constant size formula on a few bits. To summarize, *SAT* captures the difficulty of the whole class \mathcal{NP} . In particular, the \mathcal{P} vs. \mathcal{NP} problem can now be phrased as a question about the complexity of *one* problem, instead of infinitely many.

Corollary 2.14. $\mathcal{P} = \mathcal{NP}$ iff $SAT \in \mathcal{P}$.

A great advantage of having one complete problem at hand (like *SAT*), is that now, to prove that another problem (say $D \in \mathcal{NP}$) is \mathcal{NP} -complete, we only need to

design a reduction from SAT to D (namely prove $SAT \leq D$). We already know that for every $C \in \mathcal{NP}$ we have $C \leq SAT$, and transitivity of \leq takes care of the rest.

This idea was used powerfully in the next seminal paper, of Karp [66]. In his paper, he listed 21 problems from logic, graph theory, scheduling and geometry which are \mathcal{NP} -complete. This was the first demonstration of the wide spectrum of \mathcal{NP} -complete problems, and initiated an industry of finding more. A few years later Gary and Johnson [36] published their book on \mathcal{NP} -completeness, which contains hundreds of such problems from diverse branches of science, engineering and mathematics. Today thousands are known.

2.9. The nature and impact of \mathcal{NP} -completeness. It is hard to do justice to this notion in a couple of paragraphs, but we shall try. More can be found e.g. in [83].

\mathcal{NP} -completeness is a unique scientific discovery – there seems to be no parallel scientific notion which so pervaded so many fields of science and technology. It became a standard for hardness for problems whose difficulty we have yet no means of proving. It has been used both technically and allegorically to illustrate a difficulty or failure to understand natural objects and phenomena. Consequently, it has been used as a justification for channeling effort in less ambitious (but more productive) directions. We elaborate below on this effect within mathematics.

\mathcal{NP} -completeness has been an extremely flexible and extendible notion, allowing numerous variants which enabled capturing universality in other (mainly computational, but not only) contexts. It led to the ability of defining whole classes of problems by single, universal ones, with the benefits mentioned above. Much of the whole evolution of computational complexity, the theory of algorithms and most other areas in theoretical computer science have been guided by the powerful approach of reduction and completeness.

It would be extremely interesting to explain the ubiquity of \mathcal{NP} -completeness. Being highly speculative for a moment, we can make the following analogies of its mystery with physics. The existence of \mathcal{NP} -completeness in such diverse fields of inquiry may be likened to the existence of the same building blocks of matter in remote galaxies, begging for a common explanation of the same nature as the *big bang* theory. We later discuss the near lack of natural objects in the (seemingly huge) void of problems in \mathcal{NP} which are neither in \mathcal{P} nor \mathcal{NP} -complete. This raises wonders about possible “dark matter”, which we have not developed the means of observing yet.

2.10. Some \mathcal{NP} -complete problems. Again, we note that all \mathcal{NP} -complete problems are equivalent in a very strong sense. Any algorithm solving one can be simply translated into an equally efficient algorithm solving any other. We can finally see the proof of Theorem 2.1 from the beginning of this section. It follows from the following two theorems.

Theorem 2.15 ([1]). *The set 2DIO is \mathcal{NP} -complete.*

Theorem 2.16 ([2]). *The set KNOT is \mathcal{NP} -complete.*

Recall that to prove \mathcal{NP} -completeness of a set, one has to prove two things: that it is in \mathcal{NP} , and that it is \mathcal{NP} -hard. In almost all \mathcal{NP} -complete problems, membership in \mathcal{NP} (namely the existence of short certificates) is easy to prove. E.g. for *2DIO* one can easily see that if there is a positive integer solution to $Ax^2 + By + C = 0$ then indeed there is one whose length (in bits) is polynomial in the lengths of A , B , C and so a short witness is simply a root. But *KNOT* is an exception, and the short witnesses for the knot having a small genus requires Haken's algorithmic theory of normal surfaces, considerably enhanced (even short certificates for unknottedness in \mathbb{R}^3 are hard to obtain, see [53]). Let us discuss what these \mathcal{NP} -completeness results mean, first about the relationship between the two, and then about each individually.

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

But regardless of the meaning of the connection between these two problems, there is no doubt what their individual \mathcal{NP} -completeness means. Both are mathematically “nasty”, as both embed in them the full power of \mathcal{NP} . If $\mathcal{P} \neq \mathcal{NP}$, there are no efficient algorithms to describe the objects at hand. Moreover, assuming the stronger $\mathcal{NP} \neq \text{co}\mathcal{NP}$, we should not even expect complete characterization (e.g. above we should not expect short certificates that a given quadratic equation *does not* have a positive integer root).

In short, \mathcal{NP} -completeness suggests that we lower our expectations of fully understanding these properties, and study perhaps important special cases, variants etc. Note that such reaction of mathematicians may anyway follow the frustration of unsuccessful attempts at general understanding. However, the stamp of \mathcal{NP} -completeness *may* serve as moral justification for this reaction. We stress the word *may*, as the judges for accepting such a stamp can only be the mathematicians working on the problem, and how well the associated \mathcal{NP} -completeness result captures the structure they try to reveal. We merely point out the usefulness of a formal stamp of difficulty (as opposed to a general feeling), and its algorithmic meaning.

We now list a few more \mathcal{NP} -complete problems of different nature, to give a feeling for the breadth of this phenomena. Some appear already in Karp's original article [66]. Again, hundreds more can be found in [36].

- **3Color.** Given a graph, can its vertices be colored from {Red, Green, Blue} with no adjacent vertices receiving the same color?
- **Knapsack.** Given a sequence of integers a_1, \dots, a_n and b , decide if there exists a subset J such that $\sum_{i \in J} a_i = b$.
- **Integer programming.** Given a polytope in \mathbb{R}^n (by its bounding hyperplanes), does it contain an integer point?
- **Clique.** Given a graph and an integer k , are there k vertices with all pairs mutually adjacent?
- **Quadratic equations.** Given a system of multivariate polynomial equations of degree at most 2, over a *finite field* (say $\text{GF}(2)$), do they have a common root?
- **Shortest lattice vector.** Given a lattice L in \mathbb{R}^n and an integer k , is the shortest nonzero vector of L of (Euclidean) length $\leq k$?

2.11. Other problems in \mathcal{NP} (and outside it). We have seen that \mathcal{NP} contains a vast number of problems, but that difficulty-wise they seem to fall into two equivalence classes. \mathcal{P} , which are all efficiently solvable, and \mathcal{NP} -complete. Of course, if $\mathcal{P} = \mathcal{NP}$ the two classes are the same. But assuming $\mathcal{P} \neq \mathcal{NP}$, is there anything else? Ladner [71] proved the following result:

Theorem 2.17 ([71]). *If $\mathcal{P} \neq \mathcal{NP}$, then there are infinitely many levels of difficulty in \mathcal{NP} . More precisely, there are sets C_1, C_2, \dots in \mathcal{NP} such that for all i we have $C_i \leq C_{i+1}$ but $C_{i+1} \not\leq C_i$.*

But are there *natural* problems which do not fall in the main two classes \mathcal{P} and \mathcal{NP} -complete? We know only of very precious few: those on the list below, and a handful of others.

- **Integer factoring.** Given an integer, find its prime factors.
- **Approx shortest lattice vector.** Given a lattice L in \mathbb{R}^n and an integer k , does the shortest vector of L has (Euclidean) length in the range $[k, kn]$.
- **Stochastic games.** White, Black and Nature alternate moving a token on the edges of directed graph. Nature's moves are random. Given a graph, a start and target nodes for the token, does White have a strategy which guarantees that the token reach the target with probability $\geq 1/2$?
- **Graph isomorphism.** Given two graphs, are they isomorphic? Namely, is there a bijection between their vertices which preserves the edges?

Clearly, we cannot rule out that efficient algorithms will be found for any of them. But we do know that the first three are in $\mathcal{NP} \cap \text{co}\mathcal{NP}$. Thus assuming $\mathcal{NP} \neq \text{co}\mathcal{NP}$

they cannot be \mathcal{NP} -complete! A similar conclusion holds for the fourth problem, which follows from the “interactive proof” for graph non-isomorphism in Section 6.

Finding other natural examples (or better yet, classes of examples) like these will enhance our understanding of the gap $\mathcal{NP} \setminus \mathcal{P}$. Considering the examples above, we expect that mathematics is a more likely source for them than, say, industry.

2.11.1. Average-case complexity and one-way functions. It is important to mention that the “worst-case” analysis we adopted throughout (looking at the time to solve the worst input of each length) is certainly not the only interesting complexity measure. Often “average-case” analysis, focusing on typical complexity, is far more interesting to study. After all, solving a hard problem most of the time may suffice in some applications. Algorithms for natural problems under natural input distributions is an important field. But typically the input distribution is unknown, and defining “average-case” complexity in a meaningful way is highly nontrivial. This was first done by Levin [75], and the reader can find more in [57], [38].

There are also situations where *hardness* on average is crucial, as is the case in cryptography¹¹. This important field, which led to the enormous growth of electronic commerce, grew out of computational complexity, and relies on such computational assumptions. We now turn to explain the main one briefly, and recommend [39], [40] for much more.

The most basic primitive of modern cryptography is the *one-way* function, which was first defined in the seminal paper of Diffie and Hellman [29]. Intuitively, these are functions which are easy to compute (on every input) but are hard *on average* to invert. More precisely:

Definition 2.18 (One-way function). A function $f : I \rightarrow I$ is called *one-way* if $f \in \mathcal{P}$, but for any efficient algorithm A and every n ,

$$\Pr[f(A(f(x))) = x] < 1/3$$

where the probability is taken over the uniform distribution of n -bit sequences.

It is not hard to see that decision versions of one-way functions are in $\mathcal{NP} \cap \text{co}\mathcal{NP}$, and thus establishing their existence is harder than proving Conjecture 2.9. Thus cryptography postulates their existence.

Remarkably, only a handful of candidates are known today. The most popular are the multiplication of two primes (whose inverse is Integer Factorization), the exponentiation modulo a prime number (whose inverse is Discrete Logarithm)¹², and a certain linear operator (whose inverse gives the shortest vector in a lattice problem) [4]. We note also that [76] constructed a *complete* one-way function, namely a function which is one way if one-way functions exist at all.

¹¹The fact that hardness is useful at all is a surprising fact, and we shall meet it again when discussing pseudorandomness.

¹²This problem has a variant over elliptic curves.

We conclude with noting that one-way functions suffice only for some problems in cryptography, and a seemingly more powerful primitive is the so called *trap-door function*. We recommend the article [57] which deals with the relative strength of such hardness assumptions discussed in this section, and the worlds they “imply”. One basic problem is the following.

Open Problem 2.19. Does $\mathcal{P} \neq \mathcal{NP}$ imply that one-way functions exists?

2.11.2. Other types of computational problems. There are many other types of computational problems which arise naturally and do not fall into the class \mathcal{NP} . By far the most natural types are

- **Optimization problems.** Fix an \mathcal{NP} problem, and a cost function on solutions (witnesses). Given an input, find the *best* solution for it (e.g find the largest clique, the shortest path, the minimum energy configuration, etc.).
- **Counting problems.** Fix an \mathcal{NP} problem. Given an input, find the *number* of solutions (witnesses) for it. Many problems in enumerative combinatorics and in statistical physics fall in this category.
- **Strategic problems.** Given a game, find an optimal strategy for a player. Equivalently, given a position in the game, find the best move. Many problems in economics, decision theory as well as Chess and Go fall in this category.
- **Total functions.** Finding objects which are guaranteed to exist (like local optima, fixed points, Nash equilibria), usually by nonconstructive arguments [82].

We shall not elaborate on these families of important problems here. We only remark that the methodology of efficient reductions and completeness illuminate much of their computational complexity. They all fit in natural *complexity classes* like \mathcal{NP} , have complete problems, and are related in different ways to each other and to the \mathcal{P} vs. \mathcal{NP} problem.

3. Lower bounds, and attacks on \mathcal{P} vs. \mathcal{NP}

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

3.1. Diagonalization and relativization. The diagonalization technique goes back to Cantor and his argument that there are more real numbers than algebraic numbers. It was used by Gödel in his Incompleteness Theorem, and by Turing in his undecidability results, and then refined to prove computational complexity lower bounds. A typical theorem in this area is that more time buys more computational power, e.g. there are functions computable in time n^3 , say, which are not computable in time n^2 . The heart of such arguments is the existence of a “universal algorithm”, which can simulate every other algorithm with only small loss in efficiency.

Can such arguments be used to separate \mathcal{P} from \mathcal{NP} ? This depends on what we mean by “such arguments”. The paper by Baker, Gill and Solovay [12] suggested a feature shared by many similar complexity results, called *relativization*, and then proceeded to show that relativizing arguments do not suffice to resolve the \mathcal{P} vs. \mathcal{NP} question. In the three decades since that paper complexity theory grew far more sophisticated, but nevertheless almost all new results obtained do relativize (one of the few exceptions is in [110]). However, a few exceptions in the areas of probabilistic proofs (e.g. Theorem 6.3) are known not to relativize, but these are essentially *upper bound* results. More on this subject can be found in Chapter 14.3 in [81], Chapter 9.2 of [101], and even more in [34].

3.2. Boolean circuits. A Boolean circuit may be viewed as the “hardware analog” of an algorithm (software). Computation on the binary input sequence proceeds by a sequence of Boolean operations (called *gates*) from the set $\{\wedge, \vee, \neg\}$ (logical AND, OR and NEGATION) to compute the output(s). We assume that \wedge, \vee are applied to two arguments. We note that while an algorithm can handle inputs of any length, a circuit can only handle one input length (the number of input “wires” it has). A circuit is commonly represented as a (directed, acyclic) graph, with the assignments of gates to its internal vertices. We note that a Boolean formula is simply a circuit whose graph structure is a tree.

Recall that I denotes the set of all binary sequences, and that I_k is the set of sequences of length exactly k . If a circuit has n inputs and m outputs, it is clear that it computes a function $f: I_n \rightarrow I_m$. The efficiency of a circuit is measured by its *size*, which is the analog of time in algorithms.

Definition 3.1 (Circuit size). Denote by $S(f)$ the size of the smallest Boolean circuit computing f .

As we care about asymptotic behavior, we shall be interested in sequences of functions $f = \{f_n\}$, where f_n is a function on n input bits. We shall study the complexity $S(f_n)$ asymptotically as a function of n , and denote it $S(f)$. E.g. let PAR be the parity function, computing if the number of 1’s in a binary string is even or odd. Then PAR_n is its restriction to n -bit inputs, and $S(PAR) = O(n)$.¹³

¹³We use standard asymptotic notation. For integer functions g, h we write $g = O(h)$ (as well as $h = \Omega(g)$) if for some fixed constant $C > 0$ all integers n satisfy $g(n)/h(n) \leq C$. We write $g = o(h)$ if $g(n)/h(n)$ tends to 0 as n tends to infinity.

It is not hard to see that an algorithm (say a Turing machine) for a function f that runs in time T gives rise to a circuit family for the functions f_n of sizes (respectively) $(T(n))^2$, and so efficiency is preserved when moving from algorithms to circuits. Thus proving lower bounds for circuits implies lower bounds for algorithms, and we can try to attack the \mathcal{P} vs. \mathcal{NP} this way.

Definition 3.2 (The class \mathcal{P}/poly). Let \mathcal{P}/poly denote the set of all functions computable by a family of polynomial size circuits.

Conjecture 3.3. $\mathcal{NP} \not\subseteq \mathcal{P}/\text{poly}$.

Is this a reasonable conjecture? As mentioned above, $\mathcal{P} \subseteq \mathcal{P}/\text{poly}$. Does the converse hold? It actually fails badly! There exist undecidable functions f (which cannot be computed by Turing machines at all, regardless of their running time), that have linear-size circuits. This extra power comes from the fact that circuits for different input lengths share no common description (and thus this model is sometimes called “non-uniform”).

So is not proving circuit lower bounds a much harder task than proving $\mathcal{P} \neq \mathcal{NP}$ question? There is a strong sentiment that the extra power provided by non-uniformity is irrelevant to \mathcal{P} vs. \mathcal{NP} . This sentiment comes from a result of Karp and Lipton [67], proving that $\mathcal{NP} \subseteq \mathcal{P}/\text{poly}$ implies a surprising uniform “collapse”, similar to, but weaker than the statement $\mathcal{NP} = \text{co}\mathcal{NP}$.

Still, what motivates replacing the Turing machine by the potentially more powerful circuit families, when seeking lower bounds? The hope is that focusing on a *finite* model will allow for combinatorial techniques to analyze the power and limitations of efficient algorithms. This hope has materialized in the study of restricted classes of circuits (see e.g. Section 3.2.2).

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

This is due to the gap between the number of functions and the number of small circuits. Fix the number of inputs bits n . The number of possible functions on n bits is precisely 2^{2^n} . On the other hand, the number of circuits of size s is (via a crudely estimating the number of graphs of that size) at most 2^{s^2} . Since every circuit computes one function, we must have $s > 2^{n/3}$ for *most* functions.

Theorem 3.4. *For almost every function $f: I_n \rightarrow \{0, 1\}$, $\mathcal{S}(f) \geq 2^{n/3}$.*

So hard functions for circuits (and hence for Turing machines) abound. However, the hardness above is proved via a counting argument, and thus supplies no way of putting a finger on one hard function. We shall return to the nonconstructive nature of this problem in Section 4. So far, we cannot prove such hardness for any *explicit* function f (e.g., for an \mathcal{NP} -complete function like *SAT*).

Conjecture 3.5. $\mathbf{S}(\text{SAT}) \neq 2^{o(n)}$.

The situation is even worse – no *nontrivial* lower-bound is known for any explicit function. Note that for any function f on n bits (which depends on all its inputs), we trivially must have $\mathbf{S}(f) \geq n$, just to read the inputs. The main open problem of circuit complexity is beating this trivial bound.

Open Problem 3.6. Find an explicit function $f: I_n \rightarrow I_n$ for which $\mathbf{S}(f) \neq O(n)$.

A particularly basic special case of this problem, is the question whether addition is easier to perform than multiplication. Let *ADD* and *MULT* denote, respectively, the addition and multiplication functions on a pair of integers (presented in binary). For addition we have an optimal upper bound; that is, $\mathbf{S}(\text{ADD}) = O(n)$. For multiplication, the standard (elementary school) quadratic-time algorithm can be greatly improved [96] (via Discrete Fourier Transforms) to slightly super-linear, yielding $\mathbf{S}(\text{MULT}) = O(n \log n \log \log n)$. Now, the question is *whether or not there exist linear-size circuits for multiplication* (i.e., is $\mathbf{S}(\text{MULT}) = O(n)$)?

Unable to prove any nontrivial lower bound, we now turn to restricted models. There has been some remarkable successes in developing techniques for proving strong lower bounds for natural restricted classes of circuits. We discuss in some detail only one such model.

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

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

A counting argument similar to the one we used for general circuits, shows that most monotone functions require exponential size monotone circuits. Still, proving a super-polynomial lower bound on an explicit monotone function was open for over 40 years, till the invention of the so-called *approximation method* by Razborov [89].

Theorem 3.7 ([89], [6]). *CLIQUE requires exponential size monotone circuits.*

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

1. Replacing one particular gate by its approximator can only affect the output of the circuit on very few (in some natural but nontrivial counting measure) inputs. Thus in a small circuit, having a few gates, even replacing all gates results in a circuit that behaves as the original circuit on most inputs.
2. However, the output of *every* circuit (regardless of size) made of the approximating gates, produces a function which disagrees with *CLIQUE* on many inputs.

The *CLIQUE* function is well known to be \mathcal{NP} -complete, and it is natural to wonder if small monotone circuits suffice for monotone functions in \mathcal{P} . However, the approximation method was also used by Razborov [90] to prove a super polynomial size lower bound for monotone circuits computing the *Perfect Matching* problem (which is monotone and is in \mathcal{P}): given a graph, can one pair up the vertices such that every pair is connected by an edge?

Theorem 3.8 ([90]). *Perfect Matching requires super polynomial size monotone circuits.*

Interestingly, no exponential lower bound is known for monotone circuits for this problem, but different techniques [88] prove that it requires exponential size monotone *formulae* (namely circuits which are trees), and [107] gives exponential size monotone circuit lower bounds for another natural problem in \mathcal{P} .

3.2.3. Why is it hard to prove circuit lower bounds? The 1980s have seen a flurry of new techniques for proving circuit lower bounds on natural, restricted classes of circuits. Besides the *Approximation Method*, these include the *Random Restriction* method of Furst, Saxe, Sipser [35] and Ajtai [5] (used to prove lower bounds on constant depth circuits), the *Communication Complexity* method of Karchmer and Wigderson [64] (used to prove lower bounds on monotone formulae), and others (see the survey [18]). But they all fall short of obtaining any nontrivial lower bounds for general circuits, and in particular proving that $\mathcal{P} \neq \mathcal{NP}$.

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

Remarkably, complexity theory can make this vague statement into a theorem! Thus we have a “formal excuse” for our failure so far: we can classify a general set of ideas and tools, which are responsible for virtually all restricted lower bounds known, yet must necessarily fail for proving general ones. This introspective result, developed by Razborov and Rudich [93], suggests a framework called *Natural Proofs*. Very briefly, a lower bound proof is *natural*, if it applies to a *large, easily recognizable* set of functions. They first show that this framework encapsulates *all known* lower bounds. Then they show that natural proofs of general circuit lower bounds are

unlikely, in the following sense. Any natural proof of a lower bound surprisingly implies, as a side-effect, subexponential algorithms for inverting *every* candidate one-way function.

Specifically, a *natural* (in this formal sense) lower bound would imply subexponential algorithms for such functions as Integer Factoring and Discrete Logarithm, generally believed to be difficult (to the extent that the security of electronic commerce worldwide relies on such assumptions). This connection strongly uses *pseudorandomness* which will be discussed later. A simple corollary is that no natural proof exists to show that integer factoring requires circuits of size $2^{n^{1/100}}$ (the best current upper bound is $2^{n^{1/3}}$).

One interpretation of the aforementioned result, is an “independence result” of general circuit lower bounds from a certain natural fragment of Peano arithmetic. This may suggest that the \mathcal{P} vs. \mathcal{NP} problem may be independent from Peano arithmetic, or even set theory, which is certainly a possibility.

We finally note that it has been over 10 years since the publication of the Natural Proof paper. The challenge it raised: *prove a non natural lower bound* was not yet met!

4. Proof complexity

For extensive surveys on this material see [13] and in [95].

The concept of *proof* is what distinguishes the study of mathematics from all other fields of human inquiry. Mathematicians have gathered millennia of experience to attribute such adjectives to proofs as “insightful, original, deep” and most notably, “difficult”. Can one quantify, mathematically, the difficulty of proving various theorems? This is exactly the task undertaken in proof complexity. It seeks to classify theorems according to the difficulty of proving them, much like circuit complexity seeks to classify functions according to the difficulty of computing them. In proofs, just like in computation, there will be a number of models, called *proof systems* capturing the power of reasoning allowed to the prover.

Proof systems abound in all areas of mathematics (and not just in logic). Let us see some examples.

1. Hilbert’s Nullstellensatz is a (sound and complete) proof system in which *theorems* are inconsistent sets of polynomial equations. A *proof* expresses the constant 1 as a linear combination of the given polynomials.
2. Each finitely presented group can be viewed as a proof system, in which *theorems* are words that reduce to the identity element. A *proof* is the sequence of substituting relations to generate the identity.
3. Reidemeister moves are a proof system in which *theorems* are trivial, unknotted, knots. A *proof* is the sequences of moves reducing the given plane diagram of the knot into one with no crossings.

4. von Neumann’s Minimax theorem gives a proof system for every zero-sum game. A *theorem* is an optimal strategy for White, and its *proof* is a strategy for Black with the same value.

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

1. In the Nullstellensatz (over fields of characteristic 0), length (of the “coefficient” polynomials, measured usually by their degree and height) usually plays a crucial role in the efficiency of commutative algebra software, e.g. Gröbner basis algorithms.
2. The word problem in general is undecidable. For hyperbolic groups, Gromov’s polynomial upper bound on proof length has many uses, perhaps the most recent is in his own construction of finitely presented groups with no uniform embeddings into Hilbert space [48]
3. Reidemeister moves are convenient combinatorially, but the best upper bounds on length in this system to prove that a given knot is unknotted are exponential [52]. Stronger proof systems were developed to give polynomial upper bounds for proving unknottedness [53].
4. In zero-sum games, happily all proofs are of linear size.

We stress that the asymptotic view point – considering *families* of “theorems” and measuring their proof length as a function of the description length of the theorems – is natural and prevalent. As for computation, this asymptotic viewpoint reveals structure of the underlying mathematical objects, and economy (or efficiency) of proof length often means a better understanding. While this viewpoint is appropriate for a large chunk of mathematical work, you may rebel that it cannot help explaining the difficulty of *single* problems, such as the Riemann Hypothesis or \mathcal{P} vs. \mathcal{NP} . But even such theorems may be viewed asymptotically (not always illuminating them better though). The Riemann Hypothesis has equivalent formulations as a sequence of finite statements, e.g. about cancellations in the Möbius function. More interestingly, we shall see later a formulation of \mathcal{P}/poly vs. \mathcal{NP} problem, as a sequence of finite statements which are strongly related to the Natural Proofs paradigm mentioned above.

All theorems which will concern us in this section are *universal* statements (e.g. an inconsistent set of polynomial equations is the statement that *every* assignments to the variables fails to satisfy them). A short proof for a universal statement constitutes an equivalent formulation which is *existential* – the existence of the proof itself (e.g. the existence of the “coefficient” polynomials in Nullstellensatz which implies this inconsistency). The mathematical motivation for this focus is clear – the ability to describe a property both universally and existentially constitutes *necessary and sufficient* conditions – a holy grail of mathematical understanding. Here we shall be picky and quantify that understanding according to our usual computational yardstick – the *length* of the existential certificate.

We shall restrict ourselves to *propositional* tautologies. This will automatically give an exponential (thus a known, finite) upper bound on the proof length, and will restrict the ballpark (as with \mathcal{P} vs. \mathcal{NP}) to the range between polynomial and exponential. The type of statements, theorems and proofs we shall deal with is best illustrated by the following example.

4.1. The pigeonhole principle – a motivating example. Consider the well-known “pigeonhole principle”, stating that there is no injective mapping from a finite set to a smaller one. While trivial, we note that this principle was essential for the counting argument proving the *existence* of exponentially hard functions (Theorem 3.4) – this partially explains our interest in its proof complexity. More generally, this principle epitomizes *non-constructive* arguments in mathematics, such as Minkowski’s theorem that a centrally symmetric convex body of sufficient volume must contain a lattice point. In both results, the proof does not provide any information about the object proved to exist. We note that other natural tautologies capture the combinatorial essence of topological proofs (e.g. Brauer’s fixed point theorem, the Borsuk–Ulam theorem and Nash’s equilibrium) – see [82] for more.

Let us formulate it and discuss the complexity of proving it. First, we turn it into a sequence of finite statements. Fix $m > n$. Let PHP_n^m stand for the statement *there is no 1-1 mapping of m pigeons to n holes*. To formulate it mathematically, imagine an $m \times n$ matrix of Boolean variables x_{ij} describing a hypothetical mapping (with the interpretation that $x_{ij} = 1$ means that the i th pigeon is mapped to the j th hole¹⁴).

Definition 4.1 (The pigeonhole principle). The pigeonhole principle PHP_n^m now states that

- either pigeon i is not mapped anywhere (namely, *all* x_{ij} for a fixed i are zeros),
- or that some two are mapped to the same hole (namely, for some different i, i' and some j we have $x_{ij} = x_{i'j} = 1$).

These conditions are easily expressible as a formula in the variables x_{ij} (called *propositional formula*), and the pigeonhole principle is the statement that this formula is a *tautology* (namely satisfied by *every* truth assignment to the variables).

Even more conveniently, the negation of this tautology (which is a *contradiction*) can be captured by a collection of constraints on these Boolean variables which are mutually contradictory. These constraints can easily be written in different languages:

- **Algebraic:** as a set of constant degree polynomials over $\text{GF}(2)$.
- **Geometric:** as a set of linear inequalities with integer coefficients (to which we seek a $\{0, 1\}$ solution).
- **Logical:** as a set of Boolean formulae.

¹⁴Note that we do not rule out the possibility that some pigeon is mapped to more than one hole – this condition can be added, but the truth of the principle remains valid without it.

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

4.2. Propositional proof systems and \mathcal{NP} vs. $\text{co}\mathcal{NP}$. Most definitions and results in this subsection come from the paper which initiated this research direction, by Cook and Reckhow [26]. We define proof systems and the complexity measure of proof length for each, and then relate these to complexity questions we have met already.

All theorems we shall consider will be propositional tautologies. Here are the salient features that we expect¹⁵ from any proof system.

- **Completeness.** Every true statement has a proof.
- **Soundness.** No false statement has a proof.
- **Verification efficiency.** Given a mathematical statement T and a purported proof π for it, it can be easily checked if indeed π proves T in the system. Note that here efficiency of the verification procedure refers to its running-time measured in terms of the *total length of the alleged theorem and proof*.

Remark 4.2. Note that we dropped the requirement used in the definition of \mathcal{NP} , limiting the proof to be short (polynomial in the length of the claim). The reason is, of course, that proof length is our measure of complexity.

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

Definition 4.3 (Proof systems, [26]). A (*propositional*) *proof system* is a polynomial-time Turing machine M with the property that T is a tautology if and only if there exists a (“*proof*”) π such that $M(\pi, T) = 1$.¹⁶

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

Note that M_{TT} runs in polynomial time in its input length, which the combined length of formula and proof. But in the system M_{TT} proofs are (typically) of exponential length in the size of the given formula. This leads us to the definition of the

¹⁵Actually, even the first two requirements are too much to expect from strong proof systems, as Gödel famously proved in his Incompleteness Theorem. However, for propositional statements which have finite proofs there are such systems.

¹⁶In agreement with standard formalisms (see below), the proof is seen as coming before the theorem.

efficiency (or complexity) of a general propositional proof system M – how short is the shortest proof of each tautology.

Definition 4.4 (Proof length, [26]). For each tautology T , let $\mathbf{S}_M(T)$ denote the size of the shortest proof of T in M (i.e., the length of the shortest string π such that M accepts (π, T)). Let $\mathbf{S}_M(n)$ denote the maximum of $\mathbf{S}_M(T)$ over all tautologies T of length n . Finally, we call the proof system M *polynomially bounded* iff for all n we have $\mathbf{S}_M(n) = n^{O(1)}$.

Is there a polynomially bounded proof system (namely one which has polynomial size proofs for all tautologies)? The following theorem provides a basic connection of this question with computational complexity, and the major question of Section 2.5. Its proof follows quite straightforwardly from the \mathcal{NP} -completeness of *SAT*, the problem of satisfying propositional formulae (and the fact that a formula is unsatisfiable iff its negation is a tautology).

Theorem 4.5 ([26]). *There exists a polynomially bounded proof system if and only if $\mathcal{NP} = \text{co}\mathcal{NP}$.*

In the next section we focus on natural restricted proof systems. We note that a notion of reduction between proof systems, called *polynomial simulation*, was introduced in [26] and allows to create a partial order of the relative power of some systems. This is but one example to the usefulness of the methodology developed within complexity theory after the success of \mathcal{NP} -completeness.

4.3. Concrete proof systems. All proof systems in this section are of the familiar variety, starting with the deductive system introduced in *The Elements* of Euclid for plane geometry. We start with a list of formulae, and using simple (and sound!) derivation rules infer new ones (each formula is called a *line* in the proof). In the *contradiction* systems below, we start with a contradictory set of formulae, and derive a basic contradiction (e.g. $\neg x \wedge x$, $1 = 0$, $1 < 0$), depending on the setting. We highlight some results and open problems on the proof length of basic tautologies in algebraic, geometric and logical systems.

4.3.1. Algebraic proof systems. We restrict ourselves to the field $\text{GF}(2)$. Here a natural representation of a Boolean contradiction is a set of polynomials with no common root. We always add to such a collection the polynomials $x^2 - x$ (for all variables x) which ensure Boolean values (and so we can imagine that we are working over the algebraic closure).

Hilbert's Nullstellensatz suggests a proof system. If f_1, f_2, \dots, f_n (with any number of variables) have no common root, there must exist polynomials g_1, g_2, \dots, g_n such that $\sum_i f_i g_i \equiv 1$. The g_i 's constitute a proof, and we may ask how short its description is.

A related, but far more efficient system (intuitively based on computations of Gröbner bases) is Polynomial Calculus, abbreviated PC, which was introduced in [22].

The *lines* in this system are polynomials (represented explicitly by all coefficients), and it has two *deduction rules*, capturing the definition of an *ideal*: For any two polynomials g, h and variable x_i , we can use g, h to derive $g + h$, and we can use g and x_i to derive $x_i g$. It is not hard to see (using linear algebra), that if this system has a proof of length s for some tautology, then this proof can be found in time polynomial in s . Recalling our discussion on \mathcal{P} vs. \mathcal{NP} , we do not expect such a property from really strong proof systems.

The PC is known to be exponentially stronger than Nullstellensatz. More precisely, there are tautologies which require exponential length Nullstellensatz proofs, but only polynomial PC-proofs. However, strong size lower bounds (obtained from degree lower bounds) are known for PC system as well. Indeed, the pigeonhole principle is hard for this system. For its natural encoding as a contradictory set of quadratic polynomials, Razborov [91] proved

Theorem 4.6 ([91]). *For every n and every $m > n$, $S_{PC}(PHP_n^m) \geq 2^{n/2}$, over every field.*

4.3.2. Geometric proof systems. Yet another natural way to represent Boolean contradictions is by a set of regions in space containing no integer points. A wide source of interesting contradictions are Integer Programs from combinatorial optimization. Here, the constraints are (affine) linear inequalities with integer coefficients (so the regions are subsets of the Boolean cube carved out by halfspaces). A proof system infers new inequalities from old ones in a way which does not eliminate integer points.

The most basic system is called Cutting Planes (CP), introduced by Chvátal [20]. Its *lines* are linear inequalities with integer coefficients. Its *deduction rules* are (the obvious) addition of inequalities, and the (less obvious) dividing the coefficients by a constant (and rounding, taking advantage of the integrality of the solution space)¹⁷.

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

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

Theorem 4.7 ([84]). $S_{CP}(CLIQUE_n^{\sqrt{n}}) \geq 2^{n^{1/10}}$.

The proof of this theorem by Pudlak [84] is quite remarkable. It *reduces* this proof complexity lower bound into a circuit complexity lower bound. In other words, he

¹⁷E.g. from the inequality $2x + 4y \geq 1$ we may infer $x + 2y \geq \frac{1}{2}$, and by integrality, $x + 2y \geq 1$.

shows that any short CP-proof of tautologies of certain structure, yields a small circuit computing a related Boolean function. You probably guessed that for the tautology at hand, the function is indeed the *CLIQUE* function introduced earlier. Moreover, the circuits obtained are *monotone*, but of the following, very strong form. Rather than allowing only \wedge, \vee as basic gates, they allow *any* monotone binary operation on real numbers! Pudlak then goes to generalize Razborov’s approximation method (Section 3.2.2) for such circuits and proves an exponential lower bound on the size they require to compute *CLIQUE*.

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

The most basic proof system, called **Frege** system, puts no restriction on the formulae manipulated by the proof. It has one *derivation rule*, called the *cut rule*: from the two formulas $A \vee C, B \vee \neg C$ we may infer the formula $A \vee B$. Every basic book in logic has a slightly different way of describing the Frege system – one convenient outcome of the computational approach, especially the notion of efficient reductions between proof systems, is a proof (in [26]) that they are *all* equivalent, in the sense that the shortest proofs (up to polynomial factors) are independent of which variant you pick!

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

Theorem 4.8 ([19]). $S_{\text{Frege}}(\text{PHP}_n^{n+1}) = n^{O(1)}$.

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

Open Problem 4.9. Prove superpolynomial lower bounds for the Frege system.

As lower bounds for Frege are hard, we turn to subsystems of Frege which are interesting and natural. The most widely studied system is **Resolution**. Its importance stems from its use by most propositional (as well as first order) *automated theorem provers*, often called Davis–Putnam or DLL procedures [28]. This family of algorithms is designed to find proofs of Boolean tautologies, arising in diverse applications from testing computer chips or communication protocols, to basic number theory results.

The *lines* in Resolution refutations are *clauses*, namely disjunctions of literals (like $x_1 \vee x_2 \vee \neg x_3$). The *inference cut rule* simplifies to the *resolution rule*: for two clauses A , B and variable x , we can use $A \vee x$ and $B \vee \neg x$ to derive the clause $A \vee B$.

Historically, the first major result of proof complexity was Haken's¹⁸ [49] exponential lower bound on Resolution proofs for the pigeonhole principle.

Theorem 4.10 ([49]). $S_{\text{Resolution}}(\text{PHP}_n^{n+1}) = 2^{\Omega(n)}$.

To prove it, Haken developed the *bottleneck method*, which is related to both the random restriction and approximation methods mentioned in the circuit complexity chapter. This lower bound was extended to *random tautologies* (under a natural distribution) in [21]. The *width method* of [15] provides much simpler proofs for both results.

4.4. Proof complexity vs. circuit complexity. These two areas look like very different beasts, despite the syntactic similarity between the local evolution of computation and proof. To begin with, the number of objects they care about differ drastically. There are doubly exponentially number of functions (on n bits), but only exponentially many tautologies of length n . Thus a counting argument shows that some functions (albeit non explicit) require exponential circuit lower bounds (Theorem 3.4), but no similar argument can exist to show that some tautologies require exponential size proofs. So while we prefer lower bounds for natural, explicit tautologies, *existence* results of hard tautologies for strong systems are interesting in this setting as well.

Despite the different nature of the two areas, there are deep connections between them. Quite a few of the techniques used in circuit complexity, most notably *Random Restrictions* were useful for proof complexity as well. The lower bound we saw in the previous subsection is extremely intriguing: a monotone circuit lower bound directly implies a (nonmonotone) proof system lower bound! This particular type of reduction, known as the *Interpolation Method* was successfully used for other, weak, proof systems, like Resolution. It begs the question if one can use reductions of a similar nature to obtain lower bounds for strong system (like Frege), from (yet unproven) circuit lower bounds?

Open Problem 4.11. Does $\mathcal{NP} \not\subseteq \mathcal{P}/\text{poly}$ imply superpolynomial Frege lower bounds?

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

¹⁸Armin Haken, the son of Wolfgang Haken cited earlier for his work on knots.

We conclude with a tautology capturing the \mathcal{P}/poly vs. \mathcal{NP} question. Thus we use proof complexity to try showing that proving circuit lower bounds is difficult.

This tautology, suggested by Razborov, simply encodes the statement $\mathcal{NP} \not\subseteq \mathcal{P}/\text{poly}$, namely that SAT does not have small circuits. More precisely, fix n , an input size to SAT , and s , the circuit size lower bound we attempt to prove¹⁹. The variables of our “Lower Bound” formula LB_n^s encode a circuit C of size s , and the formula simply checks that C disagrees with SAT on at least one instance ϕ of length n (namely that either $\phi \in SAT$ and $C(\phi) = 0$ or $\phi \notin SAT$ and $C(\phi) = 1$.) Note that LB_n^s has size $N = 2^{O(n)}$, so we seek a superpolynomial in N lower bound on its proof length²⁰.

Proving that LB_n^s is hard for Frege will in some sense give another explanation to the difficulty of prove circuit lower bound. Such a result would be analogous to the one provided by Natural Proofs, only without relying on the existence of *one-way* functions. But paradoxically, the same inability to prove circuit lower bounds seems to prevent us from proving this proof complexity lower bound! Even proving that LB_n^s is hard for Resolution has been extremely difficult. It involves proving hardness of a *weak* pigeonhole principle²¹ – one with exponentially more pigeons than holes. It was finally achieved with the tour-de-force of Raz [87], and further strengthening of [92].

5. Randomness in computation

The marriage of randomness and computation has been one of the most fertile ideas in computer science, with a wide variety of ideas and models ranging from cryptography to computational learning theory to distributed computing. It enabled new understanding of fundamental concepts such as knowledge, secret, learning, proof, and indeed, randomness itself. In this and the next section we shall just touch the tip of the iceberg, things most closely related to the questions of efficient computation and proofs. The following two subsections tell the contradicting stories on the power and weakness of algorithmic randomness. Good sources are [79], [39] and the relevant chapters in [95].

5.1. The power of randomness in algorithms. Let us start with an example, which illustrates a potential dilemma met by mathematicians who try to prove identities. Assume we work here over the rationals \mathbb{Q} . The $n \times n$ Vandermonde matrix $V(x_1, \dots, x_n)$ in n variables has $(x_i)^{j-1}$ in the (i, j) position. The Vandermonde Identity is:

Proposition 5.1. $\det V(x_1, \dots, x_n) \equiv \prod_{i < j} (x_i - x_j)$.

While this particular identity is simple to prove, many others like it are far harder. Suppose you conjectured an identity $f(x_1, \dots, x_n) \equiv 0$, concisely expressed (as

¹⁹E.g. we may choose $s = n^{\log \log n}$ for a superpolynomial bound, or $s = 2^{n/1000}$ for an exponential one.

²⁰Of course, if $\mathcal{NP} \subseteq \mathcal{P}/\text{poly}$ then this formula is *not* a tautology, and there is no proof at all.

²¹This explicates the connection we mentioned between the pigeonhole principle and the counting argument proving existence of hard functions

above) by a short formula say, and wanted to know if it is true before investing much effort in proving it. Of course, if the number of variables n and the degree d of the polynomial f are large (as in the example), expanding the formula to check that all coefficients vanish will take exponential time and is thus infeasible. Indeed, no subexponential time algorithm for this problem is known! Is there a quick and dirty way to find out?

A natural idea suggests itself: assuming f is *not* identically zero, then the variety it defines has measure zero, and so if we pick *at random* values to the variables, chances are we shall miss it. If f is identically zero, every assignment will evaluate to zero. It turns out that the random choices can be restricted to a finite domain, and the following can be simply proved:

Proposition 5.2 ([98], [113]). *Let f be a nonzero polynomial of degree at most d in n variables. Let r_i be uniformly and independently chosen from $\{1, 2, \dots, 3d\}$. Then $\Pr[f(r_1, \dots, r_n) = 0] \leq 1/3$.*

Note that since evaluating the polynomial at any given point is easy given a formula for f , the above constitutes an efficient *probabilistic* algorithm for verifying polynomial identities. Probabilistic algorithms differ from the algorithms we have seen so far in two ways. First, they postulate the ability to toss coins and generate random bits. Second, they make errors. The beauty is, that if we are willing to accept both (and we should!), we seem to be getting far more efficient algorithms for seemingly hard problems.

The deep issue of whether randomness exists in nature has never stopped humans from assuming it anyway, for gambling, tie breaking, polls and more. A fascinating subject of how to harness seemingly unpredictable *weak sources of randomness* (such as sun spots, radioactive decay, weather, stock-market fluctuations or internet traffic) and converting them into a uniform stream of independent, unbiased coin flips, is the mathematical study of *randomness extractors* which we shall not describe here (see the excellent survey [99]). We shall postulate access of our algorithms to such perfect coin flips, and develop the theory from this assumption. We note that whatever replaces these random bits in practical implementations of probabilistic algorithms seems empirically to work pretty well.

The error seems a more serious issue – we compute to discover a *fact*, not a “maybe”. However, we do tolerate uncertainty in real life (not to mention computer hardware and software errors). Observe that the error of probabilistic algorithm is much more controllable – it can be decreased arbitrarily, with small penalty in efficiency. Assume our algorithm makes error at most $1/3$ on any input (as the one above). Then running it k times, with independent random choices each time, and taking a majority vote would reduce the error to $\exp(-k)$ on every input!

Thus we revise our notion of efficient computation to allow probabilistic algorithms with small error, and define the probabilistic analog \mathcal{BPP} (for Bounded error, Probabilistic, Polynomial time) of the class \mathcal{P} .

Definition 5.3 (The class \mathcal{BPP} , [37]). The function $f: I \rightarrow I$ is in \mathcal{BPP} if there exists a probabilistic polynomial time algorithm A , such that for every input x , $\Pr[A(x) \neq f(x)] \leq 1/3$.

Again, we stress that this probability bound is over the internal coin-tosses of the algorithm, and holds for *every* input. Moreover, replacing the error probability $1/3$ by $\exp(-|x|)$ leaves the definition unchanged (by the amplification idea above).

Probabilistic algorithms were used in statistics (for sampling) and physics (Monte Carlo methods), before computer science existed. However, their introduction into computer science in the 1970s, starting with the probabilistic primality tests of Solovay–Strassen [104] and Rabin [85], was followed by an avalanche that increased the variety and sophistication of problems amenable to such attacks tremendously – a glimpse to this scope can be obtained e.g. from the textbook [79]. We restrict ourselves here only to those which save *time*, and note that randomness seems to help save other resources as well!

We list here a few sample problems which have probabilistic polynomial time algorithms²², but for which the best known deterministic algorithms require exponential time. These are amongst the greatest achievements of this field.

- **Generating primes** ([104], [85]). Given an integer x (in binary), produce a prime in the interval $[x, 2x]$ (note that the prime number theorem guarantees that a random number in this interval is a prime with probability about $1/|x|$).
- **Polynomial factoring** ([63]). Given an arithmetic formula describing a multivariate polynomial (over a large finite field), find its irreducible factors²³
- **Permanent approximation** ([60]). Given a nonnegative real matrix, approximate its permanent²⁴ to within (say) a factor of 2.
- **Volume approximation** ([31]). Given a convex body in high dimension (e.g. a polytope given by its bounding hyperplanes), approximate its volume²⁵ to within (say) a factor of 2.

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

Open Problem 5.4. Is $\mathcal{BPP} = \mathcal{P}$?

The empirical answer is an emphatically *NO*: we have no idea in sight as to how to solve the problems above, and many others, even in subexponential time deterministically, let alone in polynomial time. However, the next subsection should change this viewpoint.

²²Strictly speaking they are not in \mathcal{BPP} as they compute relations rather than functions.

²³Note that it is not even clear that the output has a representation of polynomial length – but it does!

²⁴Unlike its relative, the determinant, which can be easily computed efficiently by Gauss elimination, the permanent is known to be $\#\mathcal{P}$ -complete (which implies \mathcal{NP} -hardness) to compute exactly.

²⁵Again, computing the volume exactly is $\#\mathcal{P}$ -complete.

5.2. The weakness of randomness in algorithms. Let us start from the end: if any of the numerous \mathcal{NP} -complete problems we saw above is *hard* then randomness is *weak*. There is a tradeoff between what the words *hard* and *weak* formally mean. To be concrete, we give perhaps the most dramatic such result of Impagliazzo and Wigderson [58].

Theorem 5.5 ([58]). *If SAT cannot be solved by circuits of size $2^{o(n)}$ then $\mathcal{BPP} = \mathcal{P}$. Moreover, SAT can be replaced in this statement by any problem which has $2^{O(n)}$ -time algorithms²⁶.*

Rephrasing, exponential circuit lower bounds on essentially any problem of interest imply that randomness can be *always* eliminated from algorithms without sacrificing efficiency (up to polynomial). Many variants of this result exist. Weakening the assumed lower bound does weaken the deterministic simulation of randomness, but leaves it highly nontrivial. For example, if $\mathcal{NP} \not\subseteq \mathcal{P}/\text{poly}$ then \mathcal{BPP} has deterministic algorithms with subexponential runtime $\exp(n^\epsilon)$ for every $\epsilon > 0$. Moreover, analogs are known where the hardness assumption is uniform (of the type $\mathcal{P} \neq \mathcal{NP}$), e.g. [59].

Note one remarkable nature of such theorems: if one computational task is hard, than another is easy!

We are now faced with deciding which of two extremely appealing beliefs to drop (as we discover that they are contradictory!). Either that natural problems (e.g. \mathcal{NP} -complete ones) cannot be solved efficiently, or that randomness is extremely powerful. Given that our intuition about the former seems far more established, we are compelled to conclude that randomness cannot significantly speed-up algorithms, and indeed $\mathcal{BPP} = \mathcal{P}$.

Conjecture 5.6. $\mathcal{BPP} = \mathcal{P}$.

We now turn to give a high level description of the ideas leading to this surprising set of results, which are generally known under the heading *Hardness vs. Randomness*²⁷. We refer the reader to the surveys in [39], [95] for more.

We are clearly after a general way of eliminating the randomness used by any (efficient!) probabilistic algorithm. Let A be such an algorithm, working on input x , and using as randomness the uniform distribution U_n on binary sequences of length n . Assume A computes a function f , and its error on any input is at most $1/3$. The idea is to “fool” A , replacing the distribution U_n by another distribution D , without A noticing it!

This leads to the key definition of *pseudorandomness* of Yao [111].

²⁶This class includes most \mathcal{NP} -complete problems, but far more complex ones, e.g. determining optimal strategies of games, not believed to be in \mathcal{NP} .

²⁷The title of Silvio Micali’s PhD thesis, who, with his advisor Manuel Blum constructed the first hardness based pseudorandom bit generator.

Definition 5.7 (Pseudorandomness, [111]). Call a distribution D *pseudorandom* if no efficient process²⁸ can “tell it apart”²⁹ from the uniform distribution U_n .

By the definition, any such distribution is as good as U_n , as A 's computation on x is and efficient process.

Remark 5.8. This definition specializes a more general one of *computational indistinguishability* between probability distributions, which originates in the landmark paper of Goldwasser and Micali [43]. This key *behavioristic* definition of randomness underlies the mathematical foundations of modern cryptography which are laid out in that paper. We also note that computational indistinguishability suggests a coarsening of the usual statistical distance (L_1 norm) between probability distributions, and we shall see its importance again in Section 6.2

Back to our derandomization task. Can we efficiently generate a pseudorandom distribution D from only very few random bits? Specifically, we would like to compute $D = G(U_m)$ where G is a deterministic polynomial time algorithm and $m \ll n$. Such functions G which produce pseudorandom distributions from short random *seeds* are called *pseudorandom generators*. With them, a deterministic simulation will only need to enumerate all possible 2^m seed values (rather than the trivial 2^n). For each such seed it will use the output of G as “randomness” for the computation of A on x , and take a majority vote. As the error of A was at most $1/3$ under U_n , and A 's output probability changes by at most $1/9$ between D and U_n , the new error is at most $4/9$, so the majority vote will correctly compute $f(x)$, for *every* x . If m gets down to $O(\log n)$, then $2^m = n^{O(1)}$, and this becomes a deterministic polynomial time algorithm.

But how can we construct such a pseudorandom generator G ? Since the definition of pseudorandomness depends on the computational limitations of the algorithm, one might hope to embed some hard function g into the workings of the generator G , and argue as follows. If an efficient process can distinguish the output of G from random, we shall turn it into an efficient algorithm for solving the (assumed hard) function g . This yields a contradiction.

Thus the heart is this conversion of hardness into pseudorandomness. The two main different methods for implementing this idea are the original generator of Blum–Micali and Yao [17], [111] (which must use “one-way” functions, see Definition 2.18, as its hard g 's), and the one by Nisan–Wigderson [80] (which can use any function g with an exponential time algorithm). We note that here the hardness required of g is of the *average-case* variety, which is either assumed in the former, or has to be obtained from *worst-case* hardness in the latter. Thus this field invents and uses new types of efficient *reductions*, translating nonstandard computational tasks (from distinguishing a random and pseudorandom distributions, to computing a function well on average, to computing it in the worst case).

²⁸This can mean an algorithm or a circuit.

²⁹E.g. produce a given output with noticeably different probability, say $1/9$.

We note that this very general line of attack may benefit from specialization. We saw that to derandomize a probabilistic algorithm all we need is a way to efficiently generate a low entropy distribution which *fools it*. But for specific, given algorithms this may be easier than for *all* of them. Indeed, careful analysis of some important probabilistic algorithms, and the way they use their randomness, has enabled making them deterministic via tailor-made generators. These success stories (of which the most dramatic is the recent deterministic primality test of [3]) actually suggest the route of probabilistic algorithms and then derandomization as a paradigm for *deterministic* algorithm design. More in the textbook [79]. Finally, we mention the remarkable result of [62] showing that derandomizing the simple probabilistic algorithm embodied in Proposition 5.2 is *equivalent* to proving certain circuit lower bounds.

We conclude by stressing that randomness remains indispensable in many fields of computer science, including cryptography, distributed computing, and – as we shall see next – probabilistic proofs.

6. Randomness in proofs²⁹

The introduction of randomness into proofs has been one of the most powerful ideas in theoretical computer science, with quite a number of unexpected consequences, and in particular new, powerful characterizations of \mathcal{NP} . In this section we summarize the main definitions and results of this research direction. Again, we refer the readers to the surveys in [61], [39], [95] and the references therein for more detail.

Let us start again with an example. Consider the graph isomorphism problem mentioned in Section 2.11: given two graphs G and H , determine if they are isomorphic. No polynomial time algorithm is known for this problem. Now assume that an infinitely powerful teacher (who in particular can solve such problems), wants to convince a limited, polynomial time student, that two graphs G, H are isomorphic. This is easy – the teacher simply provides the bijection between the vertices of the two graphs, and the student can verify that edges are preserved. This is merely a rephrasing of the fact that ISO , the set of all isomorphic pairs (G, H) , is in \mathcal{NP} . But is there a similar way for the teacher to convince the student that two given graphs are *not* isomorphic? It is not known if $ISO \in \text{co}\mathcal{NP}$, so we have no such short certificates for nonisomorphism. What can be done?

Here is an idea from [41], which allows the student and teacher more elaborate interaction, as well as coin tossing. The student challenges the teacher as follows. He (secretly) flips a coin to choose one of the two input graphs G or H . He then creates a *random* isomorphic copy K of the selected graph, by randomly permuting the vertex names (again with secret coin tosses). He then presents the teacher with

²⁹In this section we do *not* discuss the “probabilistic method”, a powerful proof *technique*. An excellent text on it is [7].

K , who is challenged to tell if K is isomorphic to G or H . Observe that if G and H are indeed non isomorphic as claimed, then the answer is unique, and the challenge can always be met (recall that the teacher has infinite computational power). If however G and H are isomorphic, *no* teacher can guess the origin of K with probability greater than $1/2$. Simply, the two distributions: random isomorphic copy of G , and random isomorphic copy of H , are *identically distributed*, and so cannot be told apart regardless of computational power. Furthermore, if the teacher succeeds in a 100 independent challenges of this type, his probability of succeeding in all when G and H are isomorphic go down to 2^{-100} , yielding an overwhelmingly convincing *interactive proof* that the graphs are indeed non isomorphic. And we note another remarkable fact: if you are worried about the need for the student to hide his coin tosses, there is another (more sophisticated) interactive proof due to [45] in which all coin tosses of the student are available to the prover!

We return to the general discussion. We have already discussed proof systems in sections 2.3 and 4. In both, the verifier that a given witness to a given claim is indeed a proof was required to be an efficient *deterministic* procedure. In the spirit of the previous section, we now relax this requirement and allow the verifier to toss coins, and err with a tiny probability.

To make the quantifiers in this definition clear, as well as allow more general interaction between the prover and the verifier, it will be convenient to view a proof system for a set S (e.g., of satisfiable formulae) as a *game* between an all-powerful prover and the (efficient, probabilistic) verifier: Both receive an input x , and the prover attempts to convince the verifier that $x \in S$. Completeness dictates that the prover succeeds for every $x \in S$. Soundness dictates that *every* prover fails for every $x \notin S$. In the definition of \mathcal{NP} , both of these conditions should hold *with probability 1* (in which case we may think of the verifier as deterministic). In probabilistic proof systems we relax this condition, and only require that soundness and completeness hold with high probability (e.g. $2/3$, as again the error can be reduced arbitrarily via iteration and majority vote). In other words, the verifier will only rarely toss coins that will cause it to mistake the truth of the assertion.

This extension of standard \mathcal{NP} proofs was suggested independently in two papers – one of Goldwasser, Micali and Rackoff [44] (whose motivation was from cryptography, in which interactions of this sort are prevalent), and the other by Babai [10] (whose motivation was to provide such interactive “certificates” for natural problems in group theory which were not known to be in $\text{co}\mathcal{NP}$). While the original definitions differed (in whether the coin tosses of the verifier are known to the prover or not), the paper of Goldwasser and Sipser [45] mentioned above showed both models equivalent.

This relaxation of proofs is not suggested as a substitute to the notion of mathematical truth. Rather, much like probabilistic algorithms, it is suggested to greatly increase the set of claims which can be efficiently proved in cases where tiny error is immaterial. As we shall see below, it turns out to yield enormous advances in computer science, while challenging our basic intuition about the very nature of proof.

We exhibit three different remarkable manifestations of that: the first shows that we can prove many more theorems, the second that we can convince others that we have a correct proof of a given theorem without revealing *anything* else about our proof, and the third that verifiers need only look at a handful of bits in a proof to be convinced of its validity.

6.1. Interactive proof systems. When the verifier is deterministic, we can always assume that the prover simply sends a single message (the purported “proof”), and based on this message the verifier decides whether to accept or reject the common input x as a member of the target set S .

When the verifier is probabilistic, *interaction* may add power. We thus consider a (randomized) interaction between the parties, which may be viewed as an “interrogation” by a persistent student, asking the teacher “tough” questions in order to be convinced of correctness. Since the verifier ought to be efficient (i.e., run in time polynomial in $|x|$), the number of such rounds of questions is bounded by a polynomial.

Definition 6.1 (The class \mathcal{IP} , [44], [10]). The class \mathcal{IP} (for Interactive Proofs) contains all sets S for which there is a verifier that accepts every $x \in S$ with probability 1 (after interacting with an adequate prover), but rejects any $x \notin S$ with probability at least $1/2$ (no matter what strategy is employed by the prover).

We have already seen the potential power of such proofs in the example of graph isomorphism above, and several others were given. But the full power of \mathcal{IP} begun to unfold only after an even stronger proof system, allowing *multiple provers*, was suggested by Ben-Or et al. [14] (motivated by cryptographic considerations). A lively account of the rapid progress is given in [11]. One milestone was showing that \mathcal{IP} proofs can be given to *every* set in $\text{co}\mathcal{NP}$ (indeed, much more, but for classes we have not defined).

Theorem 6.2 ([77]). $\text{co}\mathcal{NP} \subseteq \mathcal{IP}$.

This was shortly followed by a complete characterization of \mathcal{IP} by Shamir [100]. He proved it equivalent to \mathcal{PSPACE} , the class of sets computable with polynomial memory (and possibly exponential time). We note that this class contains problems which seem much harder than \mathcal{NP} and $\text{co}\mathcal{NP}$, e.g. finding optimal strategies of games.

Theorem 6.3 ([100]). $\mathcal{IP} = \mathcal{PSPACE}$.

We conclude by noting that this success story required the confluence and integration of ideas from different “corners” of computational complexity. A central technical tool which was developed for these results, and would play a major role in Section 6.3, is the arithmetic encoding of Boolean formulae by polynomials, and the ultra-fast verification of their properties.

6.2. Zero-knowledge proof systems. Assume you could prove the Riemann Hypothesis. You want to convince the mathematical world of your achievement, but am extremely paranoid that if you revealed the proof, someone else will claim it was his idea. Is there a way to resolve this dilemma? Hold on.

The thrust in this section is not to prove more theorems, but rather to have proofs with additional properties. Randomized and interactive verification procedures as in Section 6.1 allow the (meaningful) introduction of *zero-knowledge proofs*, which are proofs that yield nothing beyond their own validity.

Such proofs seem impossible – how can you convince anyone of anything they do not know already, without giving them any information? In mathematics, whenever we cannot prove a theorem ourselves, we feel that seeing a proof will *necessarily* teach us something we did not know!

Well, the interactive proof above, that two graphs are non isomorphic, at least suggest that in some special cases zero-knowledge proofs are possible! Note that in each round of that proof, the student knew perfectly well what the answer to his challenge was, so he learned nothing. In other words, *if* the graphs were indeed non isomorphic, he could have generated the conversation without the teacher's help! Nevertheless, the conversation convinced him that indeed the graphs were non isomorphic.

How can we define this notion formally? Extending the intuition above, we demand that on every correct claim, the verifier should be able to efficiently generate, *by himself*, (the probability distribution of) his conversation with the prover. More generally, we would be satisfied if what the verifier can generate by himself is *indistinguishable* from the actual conversation (in the same sense as pseudorandom distributions are indistinguishable from the uniform distribution 5.7).

This important definition of zero knowledge proof was suggested in the same seminal paper [44] which defined interactive proofs.

Now which theorems have zero-knowledge proofs? Well, if the verifier can determine the answer with no aid, it is trivial. Thus, any set in \mathcal{BPP} has a zero-knowledge proof, in which the prover says nothing (and the verifier decides by itself). A few examples believed outside \mathcal{BPP} like Graph Non-Isomorphism, are known to have such proofs unconditionally.

What is surprising is that if one allows the standard assumption of cryptography, namely assuming that *one-way functions* exist (see Section 2.11.1), then zero-knowledge proofs exist for *every* theorem of interest! Goldreich, Micali and Wigderson [41] proved:

Theorem 6.4 ([41]). *Assume the existence of one-way functions. Then every set in \mathcal{NP} has a zero-knowledge interactive proof.*

Here again we see the power of reductions and completeness! This theorem is proved in 2 steps. First, [41] gives a zero-knowledge proof for statements of the form *a given graph is 3-colorable*, using various structural properties of this problem. Then, it uses the \mathcal{NP} -completeness of this problem (in the strong form which allows

efficient translation of witnesses, not just instances, mentioned after Theorem 2.12) to infer that all \mathcal{NP} sets have a zero-knowledge proof.

We stand by the interpretation above of this theorem. If you proved the Riemann Hypothesis, and were nervous to reveal the proof lest the listener would rush to publish it first, you could convince him/her, beyond any reasonable doubt, that you indeed have such a proof, in a way which will reveal no information about it. Simply use the proof that 3Color is \mathcal{NP} -complete to translate the statement of the Riemann Hypothesis into the appropriate graph, translate your proof of it into the appropriate legal 3-coloring, and use the protocol of [41].

But the grand impact of this theorem is not in the above toy application. Zero-knowledge proofs are a major tool for forcing participants in cryptographic protocols to behave correctly, without compromising anyone's privacy. This is combined with the *secure evaluation* of functions [112], [42], where (completely different) reductions and completeness are again central to the proof. Together they allow for the implementation of just about any cryptographic task (for a good example for the complexity of such tasks try to imagine playing a game of poker over the telephone).

6.3. Probabilistically checkable proofs. In this section we turn to one of the deepest and most surprising discoveries on the power of probabilistic proofs, and its consequences to the limits of approximation algorithms.

We return to the non-interactive mode, in which the verifier receives a (alleged) written proof. But now we restrict its access to the proof so as to read only a small part of it (which may be randomly selected). An excellent analogy is to imagine a referee trying to decide the correctness of a long proof by sampling a few lines of the proof. It seems hopeless to detect a single “bug” unless the entire proof is read. But this intuition is valid only for the “natural” way of writing down proofs! It fails when *robust* formats of proofs are used (and, as usual, we tolerate a tiny probability of error).

Such robust proof systems are called PCPs (for *Probabilistically Checkable Proofs*). Loosely speaking, a PCP system for a set S consists of a probabilistic polynomial-time verifier having access to individual bits in a string representing the (alleged) proof³¹. The verifier tosses coins and accordingly accesses only a *constant* number of the bits in the alleged proof. It should accept every $x \in S$ with probability 1 (when given a real proof, adequately encoded), but rejects any $x \notin S$ with probability at least $1/2$ (no matter to which “alleged proof” it is given).

A long sequence of ideas and papers, surveyed in [8], [106], culminated in the “PCP theorem” of Arora et al.:

Theorem 6.5 (The PCP theorem, [9]). *Every set in \mathcal{NP} has a PCP system. Furthermore, there exists a polynomial-time procedure for converting any \mathcal{NP} -witness to the corresponding, “robust” PCP-proof.*

³¹In case of \mathcal{NP} -proofs the length of the proof is polynomial in the length of the input.

Indeed, the proof of the PCP theorem suggests a new way of writing “robust” proofs, in which any bug must “spread” all over. Equivalently, if the probability of finding a bug found in these handful of bits scanned by the verifier is small (say $< 1/10$), the theorem is correct! The remarkable PCP theorem was proved with a rather complex and technical proof, which has resisted significant simplification for over a decade. However, a conceptually different proof which is very elegant and much simpler was given last year by Dinur [30].

The reader may find a syntactic similarity between PCPs and error correcting codes. In the latter, if the probability of a bit being flipped in an encoded message is small, then the message can be correctly recovered from its noisy encoding. Indeed there are deep connections, and the cross fertilization between these two areas has been very significant.

The main impact of the PCP theorem (and its variants) is due to its connection, discovered by Feige et al. [33], to *hardness of approximation* (elaborated on in the surveys above). The PCP theorem has revolutionized our ability to argue that certain problems are not only hard to solve exactly, but even to get a rough approximation. We note that in practice, a near-optimal solution to a hard problem may be almost as good as an optimal one. But for decades, till the PCP theorem came along, we had almost no means of proving hardness of approximation results. Even with the PCP theorem, these are typically much harder to prove than standard \mathcal{NP} -completeness results. We mention two examples of the strongest such *inapproximability* results, both due to Hastad [54], [55]. Both are nearly tight, in that it is \mathcal{NP} -hard to approximate the solution by the factor given, but trivial to do so with slightly bigger factor. In both $\varepsilon > 0$ can be an arbitrarily small constant.

- **Linear equations.** Given a linear system of equations over $\text{GF}(2)$, approximate the maximum number of mutually satisfiable ones, to within a factor of $2 - \varepsilon$ (clearly, a factor 2 is trivial: a random assignment will do).
- **Clique.** Given a graph with n vertices, approximate its maximum clique size to within a factor $n^{1-\varepsilon}$ (clearly, a factor n is trivial: one vertex will do).

7. Some concrete open problems

We conclude this paper with a short list of open problems. They were all chosen so as to be free of any reference to computational models. Indeed all have simple elementary definitions, are natural and inviting. However, they all arise from attempts to prove computational lower bounds and have been open for decades. Solving any of them will represent important progress.

In all problems F is a field (of your choice – the questions are of interest for *any* field). We let $M_n(F)$ denote all $n \times n$ matrices over F and $\text{GL}_n(F)$ all invertible ones. When asking for an *explicit* matrix B , we really mean an infinite family B_n with some finite description.

7.1. Gauss elimination. For a matrix $A \in \text{GL}_n(F)$ let $G(A)$, the *Gauss elimination complexity* of A , denote the smallest number of row and column operations which transform A to a diagonal matrix.

Open Problem 7.1. Find an explicit Boolean matrix B with $G(B) \neq O(n)$.

7.2. Matrix rigidity. A matrix $A \in \text{M}_n(F)$ is (k, r) -rigid if for every matrix A' obtained from A by changing (arbitrarily) the values of at most k entries *per row*, $\text{rk}(A') \geq r$ (where rk is the rank over F).

Open Problem 7.2. Find an explicit Boolean matrix which is $(\sqrt{n}, n/100)$ -rigid.

Find an explicit Boolean matrix which is $(n/100, \sqrt{n})$ -rigid.

7.3. Permanent versus determinant. Define as usual the determinant and permanent polynomials by

$$\text{Det}_n(X) = \sum_{\sigma \in \mathcal{S}_n} \text{sgn}(\sigma) \prod_i X_{i, \sigma(i)}$$

and

$$\text{Per}_n(X) = \sum_{\sigma \in \mathcal{S}_n} \prod_i X_{i, \sigma(i)}.$$

Let $m(n)$ be the smallest value m such that Per_n is a projection of Det_m . Namely that the permanent of an $n \times n$ variable matrix X can be written as the determinant of an $m \times m$ matrix every entry of which is either a variable from X or a constant from F .

Open Problem 7.3. Prove that $m(n) \neq n^{O(1)}$.

Note that the field F cannot have characteristic 2.

7.4. Tensor rank (of matrix multiplication). For three $n \times n$ matrices of variables X, Y, Z define the trilinear form $T(X, Y, Z)$ by its action on the standard basis: for every i, j, k we have $T(X_{ij}, Y_{jk}, Z_{ki}) = 1$ and $T = 0$ on all other triples.

A rank 1 tensor is a product of linear forms (one in X , one in Y , one in Z), and the rank of a tensor is the smallest number of rank 1 tensors which add up to it.

Open Problem 7.4. Determine if the rank of T is $O(n^2)$ or not.

7.5. Generalized polynomials for determinant. The notion of tensor rank is slightly extended here to the affine case.

Let X be a set of variables. A *generalized monomial* is simply a product of affine functions over X . A *generalized polynomial* is a sum of generalized monomials.

Clearly generalized polynomials compute “normal” polynomials in $F[X]$, but sometimes they may be sparser (have fewer monomials). For a polynomial $q \in F[X]$ let $s(q)$ denote the minimum number of generalized monomials needed to express q as a generalized polynomial.

Open Problem 7.5. Prove that $s(\text{Det}_n) \neq n^{O(1)}$.

Acknowledgements. We acknowledge support from NSF grant CCR-0324906. Parts of this paper are revisions of material taken from a joint survey on computational complexity with Oded Goldreich, to be published in the Princeton “Compendium to mathematics” edited by Tim Gowers. I am grateful to the following colleagues for careful reading and comments on early versions of this manuscript: Noga Alon, Sanjeev Arora, Bernard Chazelle, Ron Fagin, Oded Goldreich, Nadia Heninger, Alex Lubotzky, Sasha Razborov, Peter Sarnak and Bill Steiger.

References

- [1] Adleman, L., and Manders, K., Computational complexity of decision problems for polynomials. *Proceedings of 16th IEEE Symposium on Foundations of Computer Science*, IEEE Comput. Soc. Press, Los Alamitos, CA, 1975, 169–177.
- [2] Agol, I., Hass, J., and Thurston, W. P., The Computational Complexity of Knot Genus and Spanning Area. *Trans. Amer. Math. Sci.* **358** (2006), 3821–3850.
- [3] Agrawal, M., Kayal, N., and Saxena, N., Primes is in \mathcal{P} . *Ann. of Math.* **160** (2) (2004), 781–793.
- [4] Ajtai, M., Generating Hard Instances of Lattice Problems. *Proceedings of the 28th annual ACM Symposium on Theory of Computing*, ACM Press, New York 1996, 99–108.
- [5] Ajtai, M., Σ_1 -formulae on finite structures. *Ann. Pure Appl. Logic* **24** (1) (1983), 1–48.
- [6] Alon, N., and Boppana R., The Monotone Circuit Complexity of Boolean Functions. *Combinatorica* **7** (1) (1987), 1–22.
- [7] Alon, N. and Spencer, J., *The Probabilistic Method*. 2nd edition, Wiley-Intersci. Ser. Discrete Math. Optim., John Wiley, New York 2000.
- [8] Arora, S., Probabilistic checking of proofs and the hardness of approximation problems. Ph.D. Thesis, UC Berkeley, 1994; revised version in http://eccc.hpi-web.de/eccc-local/ECCC-Books/sanjeev_book_readme.html
- [9] Arora, S., Lund, C., Motwani, R., Sudan, M., and Szegedy, M., Proof verification and the hardness of approximation problems. *J. ACM* **45** (3) (1998), 501–555.
- [10] Babai, L., Trading group theory for randomness. In *Proceedings of the 17th annual ACM Symposium on Theory of Computing*, ACM Press, New York 1985, 421–429.
- [11] Babai, L., E-mail and the unexpected power of interaction. In *Proceedings of the 5th Annual Conference on Structure in Complexity Theory*, IEEE Comput. Soc. Press, Los Alamitos, CA, 1990, 30–44.
- [12] Baker, T., Gill, J., and Solovay, R., Relativizations of the $P = ?NP$ question. *SIAM J. Comput.* **4** (1975), 431–442.

- [13] Beame, P., and Pitassi, T., Propositional Proof Complexity: Past, Present, and Future. *Bull. EATCS* **65** (1998), 66–89.
- [14] Ben-Or, M., Goldwasser, S., Kilian, J., Wigderson, A., Efficient Identification Schemes Using Two Prover Interactive Proofs. *Advances in Cryptography (CRYPTO 89)*, Lecture Notes in Comput. Sci. 435, Springer-Verlag, Berlin 1989, 498–506.
- [15] Ben-Sasson, E., and Wigderson, A., Short Proofs are Narrow - Resolution made Simple. *Proceedings of the 31st annual ACM Symposium on Theory of Computing*, ACM Press, New York 1999, 517–526.
- [16] Blum, L., Cucker, F., Shub, M., and Smale, S., *Complexity and Real Computation*. Springer-Verlag, 1998.
- [17] Blum, M., and Micali, S., How to generate cryptographically secure sequences of pseudorandom bits. *SIAM J. Comput.* **13** (1984), 850–864.
- [18] Boppana, R., and Sipser, M., The complexity of finite functions. In [72], 757–804.
- [19] Buss, S., Polynomial size proofs of the propositional pigeonhole principle. *J. Symbolic Logic* **52** (1987), 916–927.
- [20] Chvátal, V., Edmonds polytopes and a hierarchy of combinatorial problems. *Discrete Math.* **4** (1973), 305–337.
- [21] Chvátal, V., and Szemerédi, E., Many hard examples for resolution. *J. ACM* **35** (4) (1988), 759–768.
- [22] Clegg, M., Edmonds, J., and Impagliazzo, R., Using the Groebner Basis Algorithm to Find Proofs of Unsatisfiability. *Proceedings of the 28th annual ACM Symposium on Theory of Computing*, ACM Press, New York 1996, 174–183.
- [23] Cobham, A., The intrinsic computational difficulty of functions. In *Logic, Methodology, and Philosophy of Science*, North Holland, Amsterdam 1965, 24–30.
- [24] Cook, S. A., The Complexity of Theorem-Proving Procedures. *Proceedings of the 3rd annual ACM Symposium on Theory of Computing*, ACM Press, New York 1971, 151–158.
- [25] Cook, S. A., *The \mathcal{P} vs. \mathcal{NP} Problem*. CLAY Mathematics Foundation Millennium Problems, <http://www.claymath.org/millennium>.
- [26] Cook, S. A., and Reckhow, R. A., The Relative Efficiency of Propositional Proof Systems. *J. Symbolic Logic* **44** (1979), 36–50.
- [27] Cormen, T. H., Leiserson, C., and Rivest, R., *Introduction to Algorithms*. 2nd edition, MIT Press, Cambridge, MA; McGraw-Hill Book Co., New York 2001.
- [28] Davis, M., Logemann, G., and Loveland, D., A machine program for theorem proving. *J. ACM* **5** (7) (1962), 394–397.
- [29] Diffie, W., and Hellman, M., New directions in cryptography. *IEEE Trans. Information Theory* **22** (1976), 644–654.
- [30] Dinur, I., The PCP Theorem by gap amplification. *Proceedings of the 38th annual ACM Symposium on Theory of Computing*, ACM Press, New York 2006, 241–250.
- [31] Dyer, M., Frieze, A., and Kannan, R., A random polynomial time algorithm for approximating the volume of a convex body. *J. ACM* **38** (1) (1991), 1–17.
- [32] Edmonds, J., Paths, Trees, and Flowers. *Canad. J. Math.* **17** (1965), 449–467.
- [33] Feige, U., Goldwasser, S., Lovasz, L., Safra, S., and Szegedy, M., Interactive proofs and the hardness of approximating cliques. *J. ACM* **43** (2) (1996), 268–292.

- [34] Fortnow, L., The role of relativization in complexity theory. *Bull. EATCS* **52** (1994), 229–244.
- [35] Furst, M., Saxe, J., and Sipser, M., Parity, circuits and the polynomial time hierarchy. *Math. Systems Theory* **17** (1984), 13–27.
- [36] Garey, M. R., and Johnson, D. S., *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York 1979.
- [37] Gill, J., Computational complexity of probabilistic Turing machines. *SIAM J. Comput.* **6** (1977), 675–695.
- [38] Goldreich, O., Notes on Levin’s Theory of Average-Case Complexity. *ECCC TR97-058*, (1997).
- [39] Goldreich, O., *Modern Cryptography, Probabilistic Proofs and Pseudorandomness*. Algorithms Combin. 17, Springer-Verlag, Berlin 1999.
- [40] Goldreich, O., *Foundation of Cryptography*. I. Basic Tools; II. Basic Applications, Cambridge University Press, Cambridge 2001; 2004.
- [41] Goldreich, O., Micali, S., and Wigderson, A., Proofs that Yield Nothing but their Validity, or All Languages in NP have Zero-Knowledge Proof Systems. *J. ACM* **38** (1) (1991), 691–729.
- [42] Goldreich, O., Micali, S., and Wigderson, A., How to Play any Mental Game. *Proceedings of the 19th annual ACM Symposium on Theory of Computing*, ACM Press, New York 1987, 218–229.
- [43] Goldwasser, S., Micali, S., Probabilistic encryption. *J. Comput. System Sci.* **28**, (1984), 270–299.
- [44] Goldwasser, S., Micali, S., and Rackoff, C., The Knowledge Complexity of Interactive Proof Systems. *SIAM J. Comput.* **18** (1) (1989), 186–208.
- [45] Goldwasser, S., and Sipser, M., Private coins versus public coins in interactive proof systems. In *Randomness and Computation* (Silvio Micali, ed.), Advances in Computing Research 5, JAI Press, Inc., Greenwich, CT, 1989, 73–90.
- [46] Granville, A., It is easy to determine whether a given integer is prime. *Bull. Amer. Math. Soc.* **42** (2005), 3–38.
- [47] Gromov, M., *Hyperbolic groups*. In *Essays in Group Theory* (S. M. Gersten, ed.), Math. Sci. Res. Inst. Publ. 8, Springer-Verlag, New York 1987, 75–264.
- [48] Gromov, M., Random walk in random groups. *Geom. Funct. Anal.* **13** (1) (2003), 73–146.
- [49] Haken, A., The Intractability of Resolution. *Theor. Comput. Sci.* **39** (1985), 297–308.
- [50] Haken, W., Theorie der Normalflächen: Ein Isotopiekriterium für den Kreisknoten. *Acta Math.* **105** (1961), 245–375.
- [51] Hartmanis, J., Gödel, von Neumann and the $P = ?NP$ problem. *Bull. EATCS* **38** (1989), 101–107.
- [52] Hass, J., Lagarias, J. C., The number of Reidemeister Moves Needed for Unknotting. *J. Amer. Math. Soc.* **14** (2001), 399–428.
- [53] Hass, J., Lagarias, J. C., and Pippenger, N., The Computational Complexity of Knot and Link Problems. *J. ACM* **46** (1999), 185–211.
- [54] Håstad, J., Clique is hard to approximate within $n^{1-\epsilon}$. *Acta Math.* **182** (1999), 105–142.
- [55] Håstad, J., Some optimal inapproximability results. *J. ACM* **48** (2001), 798–859.

- [56] Hochbaum, D. (ed.), *Approximation Algorithms for NP-Hard Problems*. PWS Publishing Co., Boston, MA, 1996.
- [57] Impagliazzo, R., A personal view of average-case complexity. *Proceedings of the 10th IEEE Annual Conference on Structure in Complexity Theory*, IEEE Comput. Soc. Press, Los Alamitos, CA, 1995, 134–147.
- [58] Impagliazzo, R., and Wigderson, A., $P = BPP$ unless E has Subexponential Circuits: Derandomizing the XOR Lemma. *Proceedings of the 29th annual ACM Symposium on Theory of Computing*, ACM Press, New York 1997, 220–229.
- [59] Impagliazzo, R., and Wigderson, A., Randomness vs. Time: De-randomization under a uniform assumption. *Proceedings of the 39th Annual Symposium on Foundations of Computer Science*, IEEE Comput. Soc. Press, Los Alamitos, CA, 1998, 734–743.
- [60] Jerrum, M., Sinclair, A., Vigoda, E., A polynomial-time approximation algorithm for the permanent of a matrix with nonnegative entries. *J. ACM* **51** (4) (2004), 671–697.
- [61] Johnson, D., The Tale of the Second Prover, *J. Algorithms* **13** (3) (1992), 502–524.
- [62] Kabanets, V., Impagliazzo, R., Derandomizing Polynomial Identity Tests Means Proving Circuit Lower Bounds. *Comput. Complexity* **13** (1–2) (2004), 1–46.
- [63] Kaltofen, E. Polynomial Factorization. In *Computer Algebra: Symbolic and Algebraic Computation*, 2nd ed., Springer-Verlag, Wien, New York 1983, 95–113.
- [64] Karchmer, M., and Wigderson, A., Monotone Circuits for Connectivity require Super-Logarithmic Depth. *SIAM J. Discrete Math.* **3** (2) (1990), 255–265.
- [65] Karmarkar, N., A New Polynomial-Time Algorithm for Linear Programming. *Combinatorica* **4** (1984), 373–394.
- [66] Karp, R., Reducibility among combinatorial problems. In *Complexity of Computer Computations* (R. E. Miller and J. W. Thatcher, eds.), Plenum Press, New York 1972, 85–103.
- [67] Karp, R., and Lipton, R. J., Turing machines that take advice. *Enseign. Math.* (2) **28** (3–4) (1982), 191–209
- [68] Khachian, L., A polynomial time algorithm for linear programming. *Soviet Math. Doklady* **10** (1979), 191–194.
- [69] Kitaev, A., Shen, A., and Vyalii, M., *Classical and Quantum Computation*. Grad. Stud. Math. 47, Amer. Math. Soc., Providence, R.I., 2002.
- [70] Kushilevitz, E., and Nisan, N., *Communication Complexity*. Cambridge University Press, Cambridge 1997.
- [71] Ladner, R., On the Structure of Polynomial Time Reducibility. *J. ACM* **22** (1) (1975), 155–171.
- [72] van Leeuwen, J. (ed.), *Handbook of Theoretical Computer Science, Volume A, Algorithms and Complexity*. Elsevier Science Publishers, B.V., Amsterdam; MIT Press, Cambridge, MA, 1990.
- [73] Lenstra, A. K., Lenstra Jr., H. W., and Lovász, L. Factoring polynomials with rational coefficients. *Math. Ann.* **261** (1982), 515–534.
- [74] Levin, L. A., Universal search problems. *Probl. Peredaci Inform.* **9** (1973), 115–116; English transl. *Probl. Inf. Transm.* **9** (1973), 265–266.
- [75] Levin, L. A., Average Case Complete Problems. *SIAM J. Comput.* **15** (1) (1986), 285–286.

- [76] Levin, L. A., One-Way Functions and Pseudorandom Generators. *Combinatorica* **7** (4) (1987), 357–363.
- [77] Lund, C., Fortnow, L., Karloff, H., Nisan, N., Algebraic Methods for Interactive Proof Systems. *Proceedings of the 31th Annual Symposium on Foundations of Computer Science*, IEEE Comput. Soc. Press, Los Alamitos, CA, 1990, 2–10.
- [78] Miller, G.L., Riemann’s Hypothesis and Tests for Primality. *J. Comput. System Sci.* **13** (3) (1976), 300–317.
- [79] Motwani, R., and Raghavan, P., *Randomized Algorithms*. Cambridge University Press, Cambridge 1995.
- [80] Nisan, N., and Wigderson, A., Hardness vs. Randomness. *J. Comput. System Sci.* **49** (2) (1994), 149–167.
- [81] Papadimitriou, C. H., *Computational Complexity*. Addison Wesley, Reading, MA, 1994.
- [82] Papadimitriou, C. H., On the complexity of the parity argument and other inefficient proofs of existence. *J. Comput. System Sci.* **48** (3) (1994), 498–532.
- [83] Papadimitriou, C. H., NP-completeness: A retrospective. In *Automata, languages and programming* (ICALP’97), Lecture Notes in Comput. Sci. 1256, Springer-Verlag, Berlin 1997.
- [84] Pudlak, P., Lower bounds for resolution and cutting planes proofs and monotone computations. *J. Symbolic Logic* **62** (3) (1997), 981–998.
- [85] Rabin, M. O., Probabilistic algorithm for testing primality. *J. Number Theory* **12** (1980), 128–138.
- [86] Rabin, M., Mathematical theory of automata. In *Mathematical Aspects of Computer Science*, Proc. Sympos. Appl. Math. 19, Amer. Math. Soc., Providence, R.I., 1967, 153–175.
- [87] Raz, R., Resolution lower bounds for the weak pigeonhole principle. *J. ACM* **51** (2) (2004), 115–138.
- [88] Raz, R., and Wigderson, A., Monotone Circuits for Matching require Linear Depth. *J. ACM* **39** (1992), 736–744.
- [89] Razborov, A. A., Lower bounds for the monotone complexity of some Boolean functions. *Dokl. Akad. Nauk SSSR* **281** (4) (1985), 798–801; English transl. *Soviet Math. Doklady* **31** (1985), 354–357.
- [90] Razborov, A. A., Lower bounds of monotone complexity of the logical permanent function. *Mat. Zametki* **37** (6) (1985), 887–900; English transl. *Math. Notes* **37** (1985), 485–493.
- [91] Razborov, A. A., Lower Bounds for the Polynomial Calculus. *Comput. Complexity* **7** (4) (1998), 291–324.
- [92] Razborov, A. A., Resolution Lower Bounds for Perfect Matching Principles. *J. Comput. System Sci.* **69** (1) (2004), 3–27.
- [93] Razborov, A. A., and Rudich, S., Natural Proofs. *J. Comput. System Sci.* **55** (1) (1997), 24–35.
- [94] Robertson, N., and Seymour, P., Graph Minors I–XIII. *J. Combin. Theory B* (1983–1995).
- [95] Rudich, S., and Wigderson, A. (eds.), *Computational Complexity Theory*. IAS/Park-City Math. Ser. 10, Institute for Advanced Studies/Amer. Math. Soc., 2000.
- [96] Schönhage, A., and Strassen, V., Schnelle Multiplikation großer Zahlen. *Computing* **7** (1971), 281–292.

- [97] Schrijver, A., *Combinatorial Optimization. Polyhedra and Efficiency*. Algorithms Combin. 24, Springer-Verlag, Berlin 2003.
- [98] Schwartz, J. T., Fast probabilistic algorithms for verification of polynomial identities. *J. ACM* **27** (4) (1980), 701–717.
- [99] Shaltiel, R., Recent Developments in Explicit Constructions of Extractors. *Bull. EATCS* **77** (2002), 67–95.
- [100] Shamir, A., $IP = PSPACE$. *J. ACM* **39** (1992), 869–877.
- [101] Sipser, M., *Introduction to the Theory of Computation*. PWS Publishing Co., Boston, MA, 1997.
- [102] Sipser, M., The History and Status of the P versus NP Question. *Proceedings of the 24th annual ACM Symposium on Theory of Computing*, ACM Press, New York 1992, 603–618.
- [103] Smale, S., Mathematical Problems for the Next Century. In *Mathematics: Frontiers and Perspectives*, Amer. Math. Soc., Providence, RI, 2000, 271–294.
- [104] Solovay, R. M., and Strassen, V., A fast Monte-Carlo test for primality. *SIAM J. Comput.* **6** (1) (1977), 84–85.
- [105] Strassen, V., Algebraic Complexity Theory. In [72], 633–672.
- [106] Sudan, M., *Efficient Checking of Polynomials and Proofs and the Hardness of Approximation Problems*. ACM Distinguished Theses, Lecture Notes in Comput. Sci. 1001, Springer-Verlag, Berlin 1996.
- [107] Tardos, E., The Gap Between Monotone and Non-Monotone Circuit Complexity is Exponential. *Combinatorica* **7** (4) (1987), 141–142.
- [108] Tarski, A., *A decision method for elementary algebra and geometry*. University of California Press, 1951.
- [109] Valiant, L. G., Completeness classes in algebra. In *Proceedings of the eleventh annual ACM Symposium on Theory of Computing* (1979), 249–261.
- [110] N. V. Vinodchandran, $AM_{\text{exp}} \not\subseteq (NP \cap \text{coNP})/\text{poly}$. *Inform. Process. Lett.* **89** (2004), 43–47.
- [111] Yao, A. C., Theory and application of trapdoor functions. *Proceedings of the 23th annual IEEE Symposium on Foundations of Computer Science*, IEEE Comput. Soc. Press, Los Alamitos, CA, 1982, 80–91.
- [112] Yao, A. C., How to generate and exchange secrets. In *Proceedings of the 27th annual IEEE Symposium on Foundations of Computer Science*, IEEE Comput. Soc. Press, Los Alamitos, CA, 1986, 162–167.
- [113] Zippel, R. E., Probabilistic algorithms for sparse polynomials. In *Symbolic and algebraic computation* (EUROSCAM '79), Lecture Notes in Comput. Sci. 72, Springer-Verlag, Berlin 1979, 216–226.

School of Mathematics, Institute for Advanced Study, Princeton NJ 08540, U.S.A.

E-mail: avi@ias.edu