John Forbes Nash, Jr.
Michael Th. Rassias *Editors*

Open Problems in Mathematics

Springer

Open Problems in Mathematics

John Forbes Nash, Jr. • Michael Th. Rassias
Editors

Open Problems in Mathematics

Springer

Editors
John Forbes Nash, Jr. (Deceased)
Department of Mathematics
Princeton University
Princeton, NJ, USA

Michael Th. Rassias
Department of Mathematics
Princeton University
Princeton, NJ, USA

ETH-Zürich
Department of Mathematics
Zürich, Switzerland

ISBN 978-3-319-32160-8 ISBN 978-3-319-32162-2 (eBook)
DOI 10.1007/978-3-319-32162-2

Library of Congress Control Number: 2016941333

Printed on acid-free paper

This Springer imprint is published by Springer Nature
The registered company is Springer International Publishing AG Switzerland

Preface

John Forbes Nash, Jr. and Michael Th. Rassias

Learn from yesterday, live for today, hope for tomorrow.
The important thing is not to stop questioning.
– Albert Einstein (1879–1955)

It has become clear to the modern working mathematician that no single researcher, regardless of his knowledge, experience, and talent, is capable anymore of overviewing the major open problems and trends of mathematics in its entirety. The breadth and diversity of mathematics during the last century has witnessed an unprecedented expansion.

In 1900, when David Hilbert began his celebrated lecture delivered before the International Congress of Mathematicians in Paris, he stoically said:

> Who of us would not be glad to lift the veil behind which the future lies hidden; to cast a glance at the next advances of our science and at the secrets of its development during future centuries? What particular goals will there be toward which the leading mathematical spirits of coming generations will strive? What new methods and new facts in the wide and rich field of mathematical thought will the new centuries disclose?

Perhaps Hilbert was among the last great mathematicians who could talk about mathematics as a whole, presenting problems which covered most of its range at the time. One can claim this, not because there will be no other mathematicians of Hilbert's caliber, but because life is probably too short for one to have the opportunity to expose himself to the allness of the realm of modern mathematics. Melancholic as this thought may sound, it simultaneously creates the necessity and aspiration for intense collaboration between researchers of different disciplines. Thus, overviewing open problems in mathematics has nowadays become a task which can only be accomplished by collective efforts.

The scope of this volume is to publish invited survey papers presenting the status of some essential open problems in pure and applied mathematics, including old and new results as well as methods and techniques used toward their solution. One expository paper is devoted to each problem or constellation of related problems. The present anthology of open problems, notwithstanding the fact that it ranges over a variety of mathematical areas, does not claim by any means to be complete, as such a goal would be impossible to achieve. It is, rather, a collection of beautiful mathematical questions which were chosen for a variety of reasons. Some were chosen for their undoubtable importance and applicability, others because they constitute intriguing curiosities which remain unexplained mysteries on the basis of current knowledge and techniques, and some for more emotional reasons. Additionally, the attribute of a problem having a somewhat *vintage flavor* was also influential in our decision process.

The book chapters have been contributed by leading experts in the corresponding fields. We would like to express our deepest thanks to all of them for participating in this effort.

Princeton, NJ, USA
April, 2015

John F. Nash, Jr.

John F. Nash, Jr.

Michael Th. Rassias

[illegible]

Contents

A Farewell to “A Beautiful Mind and a Beautiful Person”

Michael Th. Rassias

Having found it very hard to resign myself to John F. Nash’s sudden and so tragic passing, I postponed writing my commemorative addendum to our jointly composed preface until this compilation of papers on open problems was almost fully ready for publication. Now that I have finally built up my courage for coming to terms with John Nash’s demise, my name, which joyfully adjoins his at the end of the above preface, now also stands sadly alone below the following bit of reminiscence from my privileged year as his collaborator and frequent companion.

It all started in September 2014, in one of the afternoon coffee/tea meetings that take place on a daily basis in the common room of Fine Hall, the building housing the Mathematics Department of Princeton University. John Nash silently entered the room, poured himself a cup of decaf coffee and then sat alone in a chair close by. That was when I first approached him and had a really pleasant chat about problems in the interplay of game theory and number theory. From that day onwards, our discussions became ever more frequent, and we eventually decided to prepare this volume *Open Problems in Mathematics*. The day we made this decision, he turned to me and said with his gentle voice, “I don’t want to be *just a name* on the cover though. I want to be really involved.” After that, we met almost daily and discussed for several hours at a time, examining a vast number of open problems in mathematics ranging over several areas. During these discussions, it became even clearer to me that his way of thinking was very different from that of almost all other mathematicians I have ever met. He was thinking in an unconventional, most creative way. His quick and distinctive mind was still shining bright in his later years.

This volume was practically almost ready before John and Alicia Nash left in May for Oslo, where he was awarded the 2015 Abel Prize from the Norwegian Academy of Science and Letters. We had even prepared the preface of this volume, which he was so much looking forward to see published. Our decision to include handwritten signatures, as well, was along the lines of the somewhat vintage flavor and style that he liked.

John Nash was planning to write a brief article on an open problem in game theory, which was the only problem we had not discussed yet. He was planning to prepare it and discuss about it after his trip to Oslo. Thus, he never got the opportunity to write it. On this note, and notwithstanding my ‘last-minute’ invitation, Professor Eric Maskin generously accepted to contribute a paper presenting an important open problem in cooperative game theory.

With this opportunity, I would also like to say just a few words about the man behind the mathematician. In the famous movie *A Beautiful Mind*, which portrayed his life, he was presented as a really combative person. It is true that in his early years he might have been, having also to battle with the demons of his illness. Being almost 60 years younger than him, I had the chance to get acquainted with his personality in his senior years. All the people who were around him, including myself, can avow that he was a truly wonderful person. Very kind and disarmingly simple, as well as modest. This is the reason why, among friends at Princeton, I used to humorously say that the movie should have been called *A Beautiful Mind and a Beautiful Person*. What was certainly true though was the dear love between John and Alicia Nash, who together faced and overcame the tremendous challenges of John Nash’s life. It is somehow a romantic tragedy that fate bound them to even leave this life together.

In history, one can say that among the mathematicians who have reached greatness, there are some—a selected few—who have gone beyond greatness to become legends. John Nash was one such legend.

The contributors of papers and myself cordially dedicate this volume to the memory and rich mathematical legacy of John F. Nash, Jr.

Princeton, NJ, USA Michael Th. Rassias

Introduction
John Nash: Theorems and Ideas

Misha Gromov

Nash was not building big theories, he did not attempt to dislodge old concepts and to promote new ones, he didn't try to be paradoxical.

Nash was solving classical mathematical problems, difficult problems, something that nobody else was able to do, not even to imagine how to do it.

His landmark theorem of 1956—one of the main achievements of mathematics of the twentieth century–reads:

> All Riemannian manifolds X can be realised as smooth submanifolds in Euclidean spaces $\mathbb{R}^q$, such that the smoothness class of the submanifold realising an X in $\mathbb{R}^q$ equals that of the Riemannian metric g on X and where the dimension q of the ambient Euclidean space can be universally bounded in terms of the dimension of X.[1]

And as far as C^1-*smooth* isometric embeddings $f : X \to \mathbb{R}^q$ are concerned, there is no constraint on the dimension of the Euclidean space except for what is dictated by the topology of X:

> Every C^1-smooth n-dimensional submanifold X_0 in $\mathbb{R}^q$ for $q \geq n+1$ can be deformed (by a C^1-isotopy) to a new C^1-position such that the induced Riemannian metric on X_0 becomes equal to a given g.[2]

At first sight, these are natural classically looking theorems. But what Nash has discovered in the course of his constructions of isomeric embeddings is far from "classical"—it is something that brings about a dramatic alteration of our understanding of the basic logic of analysis and differential geometry. Judging from

M. Gromov
IHÉS, 36 route de Chartres, 41990 Bures-sur-Yvette, France

e-mail: gromov@ihes.fr

[1]This was proven in the 1956 paper for C^r-smooth metrics, $r = 3, 4, \ldots, \infty$; the existence of *real analytic* isometric embeddings of *compact* manifolds with *real analytic* Riemannian metrics to Euclidean spaces was proven by Nash in 1966.

[2]Nash proved this in his 1954 paper for $q \geq n + 2$, where he indicated that a modification of his method would allow $q = n + 1$ as well. This was implemented in a 1955 paper by Nico Kuiper.

the classical perspective, what Nash has achieved in his papers is as impossible as the story of his life.

Prior to Nash, the following two heuristic principles, vaguely similar to the first and the second laws of thermodynamics, have been (almost?) unquestionably accepted by analysts:

1. Conservation of Regularity. *The smoothness of solutions f of a "natural" functional, in particular a differential, equation $\mathscr{D}(f) = g$ is determined by the equation itself but not by a particular class of functions f used for the proof of the existence of solutions.*
2. Increase of Irregularity. *If some amount of regularity of potential solutions f of our equations has been lost, it cannot be recaptured by any "external means," such as artificial smoothing of functions.*

Instances of the first principle can be traced to the following three Hilbert's problems:

5th: *Continuous groups are infinitely differentiable, in fact, real analytic.*
19th: *Solutions of "natural" elliptic PDE are real analytic.*
Also Hilbert's formulation of his **13th** problem on

non-representability of "interesting" functions in many variables
by superpositions of ***continuous*** *functions in fewer variables*

is motivated by this principle:

$$\textit{continuous} \Leftrightarrow \textit{real analytic}$$

as far as superpositions of functions are concerned.

Nash C^1-isometric embedding theorem shattered the conservation of regularity idea: the system of differential equations that describes isometric immersions $f : X \to \mathbb{R}^q$ may have no analytic or not even C^2-smooth solution f.

But, according to Nash's 1954 theorem, *if $q > dim(X)$, and if X is diffeomorphic, to $\mathbb{R}^n$, $n < q$, or to the n-sphere, then, no matter what Riemannian metric g you are given on this X, there are lots of isometric C^1-embeddings $X \to \mathbb{R}^q$.*[3]

Now, look at an equally incredible Nash's approach to *more regular, say C^∞-smooth, isometric embeddings*. The main lemma used by Nash, his *implicit (or inverse) function theorem*, may seem "classical" unless you read the small print:

Let $\mathscr{D} : F \to G$ be a C^∞-smooth non-linear differential operator between spaces F and G of C^∞-sections of two vector bundles over a manifold X.

[3]In the spirit of Nash but probably independently, the *continuous* $\Leftrightarrow$ *real analytic* equivalence for superpositions of functions was disproved by Kolmogorov in 1956; yet, in essence, Hilbert's **13th** problem remains unsolved: *are there algebraic (or other natural) functions in many variables that are not superpositions of* ***real analytic*** *functions in two variables?*

Also, despite an enormous progress, "true" Hilbert's 19th problem remains widely open: *what are possible singularities of solutions of elliptic PDE systems, such as minimal subvarieties and Einstein manifolds.*

If the linearization $\mathscr{L} = \mathscr{L}_{f_0}(f)$ of $\mathscr{D}$ at a point $f_0 \in F$ is invertible at $g_0 = \mathscr{D}(f_0) \in G$ by a ***differential*** *operator linear in g, say $\mathscr{M} = \mathscr{M}_{f_0}(g)$, then $\mathscr{D}$ is also invertible (by a nonlinear non-differential operator) in a (small fine) C^∞-neighborhood of $\mathscr{D}(f_0) \in G$.*

You must be a novice in analysis or a genius like Nash to believe anything like that can be ever true and/or to have a single nontrivial application.

First of all, who has ever seen inversions of differential operators again by *differential* ones?

And second of all, how on earth can $\mathscr{D}$ be inverted by means of $\mathscr{M}$ when both operators, being differential, *increase* irregularity?

But Nash writes down a simple formula for a linearized inversion $\mathscr{M}$ for the metric inducing operator $\mathscr{D}$, and he suggests a compensation for the loss of regularity by the use of *smoothing operators*.

The latter may strike you as realistic as a successful performance of perpetuum mobile with a mechanical implementation of Maxwell's demon... unless you start following Nash's computation and realize to your immense surprise that the smoothing does work in the hands of John Nash.

This, combined with a few ingenious geometric constructions, leads to C^∞-smooth isometric embeddings $f : X \to \mathbb{R}^q$ for $q = 3n^3/2 + O(n^2)$, $n = dim(X)$.

Besides the above, Nash has proved a few other great theorems, but it is his work on isometric immersions that opened a new world of mathematics that stretches in front of our eyes in yet unknown directions and still waits to be explored.

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

Scott Aaronson

Abstract In 1950, John Nash sent a remarkable letter to the National Security Agency, in which—seeking to build theoretical foundations for cryptography—he all but formulated what today we call the $\mathsf{P} \overset{?}{=} \mathsf{NP}$ problem, and consider one of the great open problems of science. Here I survey the status of this problem in 2016, for a broad audience of mathematicians, scientists, and engineers. I offer a personal perspective on what it's about, why it's important, why it's reasonable to conjecture that $\mathsf{P} \neq \mathsf{NP}$ is both true and provable, why proving it is so hard, the landscape of related problems, and crucially, what progress has been made in the last half-century toward solving those problems. The discussion of progress includes diagonalization and circuit lower bounds; the relativization, algebrization, and natural proofs barriers; and the recent works of Ryan Williams and Ketan Mulmuley, which (in different ways) hint at a duality between impossibility proofs and algorithms.

1 Introduction

Now my general conjecture is as follows: for almost all sufficiently complex types of enciphering, especially where the instructions given by different portions of the key interact complexly with each other in the determination of their ultimate effects on the enciphering, the mean key computation length increases exponentially with the length of the key, or in other words, the information content of the key ... The nature of this conjecture is such that I cannot *prove* it, even for a special type of ciphers. Nor do I expect it to be proven.—John Nash, 1950 [171]

In 1900, David Hilbert challenged mathematicians to design a "purely mechanical procedure" to determine the truth or falsehood of any mathematical statement.

For a more recent version of this survey, with numerous corrections and revisions as well as new material, the reader is invited to visit http://www.scottaaronson.com/papers/pvsnp.pdf

S. Aaronson (✉)
Department of Computer Science, University of Texas, Austin, USA
e-mail: scott@scottaaronson.com

J.F. Nash, Jr., M.Th. Rassias (eds.), *Open Problems in Mathematics*,
DOI 10.1007/978-3-319-32162-2_1

That goal turned out to be impossible. But the *question*—does such a procedure exist, and why or why not?—helped launch three related revolutions that shaped the twentieth century: one in math and science, as reasoning itself became a subject of mathematical analysis; one in philosophy, as the results of Gödel, Church, Turing, and Post showed the limits of formal systems and the power of self-reference; and one in technology, as the electronic computer achieved, not all of Hilbert's dream, but enough of it to change the daily experience of most people on earth.

The $\mathsf{P} \overset{?}{=} \mathsf{NP}$ problem is a modern refinement of Hilbert's 1900 question. The problem was explicitly posed in the early 1970s in the works of Cook and Levin, though versions were stated earlier—including by Gödel in 1956, and as we see above, by John Nash in 1950. In plain language, $\mathsf{P} \overset{?}{=} \mathsf{NP}$ asks whether there's a *fast* procedure to answer all questions *that have short answers that are easy to verify mechanically*. Here one should think of a large jigsaw puzzle with (say) 10^{1000} possible ways of arranging the pieces, or an encrypted message with a similarly huge number of possible decrypts, or an airline with astronomically many ways of scheduling its flights, or a neural network with millions of weights that can be set independently. All of these examples share two key features:

(1) a finite but exponentially-large space of possible solutions; and
(2) a fast, mechanical way to check whether any claimed solution is "valid." (For example, do the puzzle pieces now fit together in a rectangle? Does the proposed airline schedule achieve the desired profit? Does the neural network correctly classify the images in a test suite?)

The $\mathsf{P} \overset{?}{=} \mathsf{NP}$ question asks whether, under the above conditions, there's a general method to *find* a valid solution whenever one exists, and which is enormously faster than just trying all the possibilities one by one, from now till the end of the universe, like in Jorge Luis Borges' Library of Babel.

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

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

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

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

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

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

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

> If there actually were a machine with [running time] $\sim Kn$ (or even only with $\sim Kn^2$) [for some constant K independent of n], this would have consequences of the greatest magnitude. That is to say, it would clearly indicate that, despite the unsolvability of the Entscheidungsproblem, the mental effort of the mathematician in the case of yes-or-no questions could be completely [added in a footnote: apart from the postulation of axioms] replaced by machines. One would indeed have to simply select an n so large that, if the machine yields no result, there would then also be no reason to think further about the problem.

Expanding on Gödel's observation, some modern commentators have explained the importance of $\mathsf{P} \overset{?}{=} \mathsf{NP}$ as follows. It's well-known that $\mathsf{P} \overset{?}{=} \mathsf{NP}$ is one of the seven Clay Millennium Problems (alongside the Riemann Hypothesis, the Yang-Mills mass gap, etc.), for which a solution commands a million-dollar prize [66]. But even among those problems, $\mathsf{P} \overset{?}{=} \mathsf{NP}$ has a special status. For if someone discovered that $\mathsf{P} = \mathsf{NP}$, and if moreover the algorithm was efficient in practice, that person could solve not merely one Millennium Problem but *all seven* of them—for she'd simply need to program her computer to search for formal proofs of the other six conjectures.[1] Of course, if (as most computer scientists believe) $\mathsf{P} \neq \mathsf{NP}$,

[1]Here we're using the observation that, once we fix a formal system (say, first-order logic plus the axioms of ZF set theory), deciding whether a given statement has a proof at most n symbols long in that system is an NP problem, which can therefore be solved in time polynomial in n assuming

a proof of *that* would have no such world-changing implications, but even the fact that such a proof could *rule out* those implications underscores the enormity of what we're asking.

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

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

To illustrate, suppose we wanted to program a computer to create new Mozart-quality symphonies and Shakespeare-quality plays. If $\mathsf{P} = \mathsf{NP}$, and the algorithm were efficient in practice, then that really would imply that these feats could be reduced to a seemingly-easier problem, of programming a computer to *recognize* such symphonies and plays when given them. And interestingly, $\mathsf{P} = \mathsf{NP}$ might *also* help with the recognition problem: for example, by letting us train a neural network that reverse-engineered the expressed aesthetic preferences of hundreds of human experts. But how well that neural network would perform is an empirical question outside the scope of mathematics.

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

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

$\mathsf{P} = \mathsf{NP}$. We're also assuming that the other six Clay conjectures have ZF proofs that are not too enormous: say, 10^{12} symbols or fewer, depending on the exact running time of the assumed algorithm. In the case of the Poincaré Conjecture, this can almost be taken to be a fact, modulo the translation of Perelman's proof [179] into the language of ZF.

1.2.1 The Asymptotic Objection

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

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

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

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

1.2.2 The Polynomial-Time Objection

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

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

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

1.2.3 The Kitchen-Sink Objection

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

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

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

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

Certainly there are cases where this assumption is true. But there are also cases where it's false: indeed, the entire field of cryptography is about *making* the assumption false! In addition, I believe our practical experience is biased by the fact that we don't even *try* to solve search problems that we "know" are hopeless—yet that wouldn't be hopeless in a world where $\mathsf{P} = \mathsf{NP}$ (and where the algorithm was efficient in practice). For example, presumably no one would try using brute-force search to look for a formal proof of the Riemann Hypothesis one billion lines long or shorter, or a 10-megabyte program that reproduced most of the content of Wikipedia within a reasonable time (possibly needing to encode many of the principles of human intelligence in order to do so). Yet both of these are "merely" NP search problems, and things one could seriously contemplate in a world where $\mathsf{P} = \mathsf{NP}$.

1.2.4 The Mathematical Snobbery Objection

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

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

This objection might also reflect lack of familiarity with recent progress in complexity theory, which has drawn on Fourier analysis, arithmetic combinatorics, representation theory, algebraic geometry, and dozens of other subjects about which yellow books are written. Furthermore, in Sect. 6.6, we'll see Geometric Complexity Theory (GCT), a breathtakingly ambitious program for proving $\mathsf{P} \neq \mathsf{NP}$ that throws almost the entire arsenal of modern mathematics at the problem, including geometric invariant theory, plethysms, quantum groups, and Langlands-type correspondences. Regardless of whether GCT's specific conjectures pan out, they illustrate in detail how progress toward proving $\mathsf{P} \neq \mathsf{NP}$ will plausibly involve deep insights from many parts of mathematics.

1.2.5 The Sour Grapes Objection

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

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

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

1.2.6 The Obviousness Objection

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

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

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

1.2.7 The Constructivity Objection

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

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

1.3 Further Reading

There were at least four previous major survey articles about $\mathsf{P} \overset{?}{=} \mathsf{NP}$: Michael Sipser's 1992 "The History and Status of the P versus NP Question" [207]; Stephen Cook's 2000 "The P versus NPProblem" [66], which was written for the announcement of the Clay Millennium Prize; Avi Wigderson's 2006 "P,

[2]The most celebrated examples of nonconstructive proofs that algorithms exist all come from the *Robertson-Seymour graph minors theory*, one of the great achievements of twentieth-century combinatorics (for an accessible introduction, see for example Fellows [75]). The Robertson-Seymour theory typically deals with *parameterized* problems: for example, "given a graph G, decide whether G can be embedded on a sphere with k handles." In those cases, typically a fast algorithm A_k can be abstractly shown to exist for every value of k. The central problem is that each A_k requires hard-coded data—in the above example, a finite list of obstructions to the desired embedding—that no one knows how to find given k, and whose size might also grow astronomically as a function of k. On the other hand, once the finite obstruction set for a given k was known, one could then use it to solve the problem for any graph G in time $O\left(|G|^3\right)$, where the constant hidden by the big-O depended on k.

Robertson-Seymour theory also provides a few examples of non-parameterized problems that are abstractly proved to be in P but with no bound on the exponent, or abstractly proved to be $O\left(n^3\right)$ or $O\left(n^2\right)$ but with no bound on the constant. Thus, one cannot rule out the possibility that the same would happen with an NP-complete problem, and Donald Knuth [131] has explicitly speculated that $\mathsf{P} = \mathsf{NP}$ will be proven in that way. To me, however, it is unclear whether he speculates this because there is a positive reason for thinking it true, or just because it would be cool and interesting if it *was* true.

[3]As an amusing side note, there is a trick called *Levin's universal search* [141], in which one "dovetails" over all Turing machines $M_1, M_2, \ldots$ (that is, for all t, runs $M_1, \ldots, M_t$ for t steps each), halting when and if any M_i has outputs a valid solution to one's NP search problem. If we know $\mathsf{P} = \mathsf{NP}$, then we know this particular algorithm will find a valid solution, whenever one exists, in polynomial time—because clearly *some* M_i does so, and all the machines other than M_i increase the total running time by "only" a polynomial factor! With more work, one can even decrease this to a constant factor. Admittedly, however, the polynomial or constant factor will be so enormous as to negate this algorithm's practical use.

NP, and Mathematics—A Computational Complexity Perspective"[232]; and Eric Allender's 2009 "A Status Report on the P versus NP Question" [21]. All four are excellent, so it's only with trepidation that I add another entry to the crowded arena. I hope that, if nothing else, this survey shows how much has continued to occur. I cover several major topics that either didn't exist a decade ago, or existed only in much more rudimentary form: for example, the algebrization barrier, "ironic complexity theory" (including Ryan Williams's $\mathsf{NEXP} \not\subset \mathsf{ACC}$ result), the "chasm at depth three" for the permanent, and the Mulmuley-Sohoni Geometric Complexity Theory program.

The seminal papers that set up the intellectual framework for $\mathsf{P} \overset{?}{=} \mathsf{NP}$, posed it, and demonstrated its importance include those of Edmonds [74], Cobham [65], Cook [67], Karp [122], and Levin [141]. See also Trakhtenbrot [223] for a survey of Soviet thought about *perebor*, as brute-force search was referred to in Russian in the 1950s and 60s.

The classic text that introduced the wider world to P, NP, and NP-completeness, and that gave a canonical (and still-useful) list of hundreds of NP-complete problems, is Garey and Johnson [86]. Some recommended computational complexity theory textbooks—in rough order from earliest to most recent, in the material they cover—are Sipser [208], Papadimitriou [175], Schöning [199], Moore and Mertens [155], and Arora and Barak [27]. Surveys on particular aspects of complexity theory will be recommended where relevant throughout the survey.

Those seeking a nontechnical introduction to $\mathsf{P} \overset{?}{=} \mathsf{NP}$ might enjoy Lance Fortnow's charming book *The Golden Ticket* [80], or his 2009 popular article for *Communications of the ACM* [79]. My own *Quantum Computing Since Democritus* [6] gives something between a popular and a technical treatment.

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

The $\mathsf{P} \overset{?}{=} \mathsf{NP}$ problem is normally phrased in terms of *Turing machines*: a theoretical model of computation proposed by Alan Turing in 1936, which involves a one-dimensional tape divided into discrete squares, and a finite control that moves back and forth on the tape, reading and writing symbols. For a formal definition, see, e.g., Sipser [208] or Cook [66].

In this survey, I won't define Turing machines, for the simple reason that *if you know any programming language—C, Java, Python, etc.—then you already know something that's equivalent to Turing machines for our purposes.* More precisely, the *Church-Turing Thesis* holds that virtually any model of digital computation one can define will be equivalent to Turing machines, in the sense that Turing machines can simulate that model and vice versa. A modern refinement, the *Extended Church-Turing Thesis*, says that moreover, these simulations will incur at most a polynomial overhead in time and memory. If we accept this, then there's a well-defined notion of "solvable in polynomial time by a digital computer," which is independent of the low-level details of the computer's architecture: the instruction set, the rules for accessing memory, etc. This licenses us to ignore those details. The main caveats here are that

(1) the computer must be classical, discrete, and deterministic (it's not a quantum computer, an analog device, etc., nor can it call a random-number generator or any other external resource), and
(2) there must be no *a-priori* limit on how much memory the computer can address, even though any program that runs for finite time will only address a finite amount of memory.[4,5]

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

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

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

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

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

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

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

Now, P, or Polynomial-Time, is the class of all languages L for which there exists a Turing machine M that decides L in polynomial time. Also, NP, or Nondeterministic Polynomial-Time, is the class of languages L for which there exists a Turing machine M, and a polynomial p, such that for all $x \in \{0,1\}^*$,

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

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

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

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

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

2.1 NP*-Completeness*

A further concept, not part of the statement of $\mathsf{P} \overset{?}{=} \mathsf{NP}$ but central to any discussion of it, is NP*-completeness*. To explain this requires a few more definitions. An *oracle Turing machine* is a Turing machine that, at any time, can submit an instance x to an "oracle": a device that, in a single time step, returns a bit indicating whether x belongs to some given language L. An oracle that answers all queries consistently with L is called an *L-oracle*, and we write M^L to denote the (oracle) Turing machine

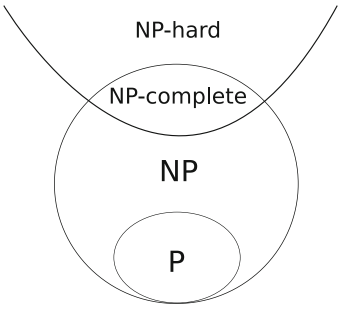

Fig. 1 P, NP, NP-hard, and NP-complete

M with L-oracle. We can then define P^L, or P *relative to* L, as the class of all languages L' for which there exists an oracle machine M such that M^L decides L' in polynomial time. If $L' \in \mathsf{P}^L$, then we also write $L' \leq_{\mathsf{P}}^{T} L$, which means "$L'$ is polynomial-time Turing-reducible to L." Note that polynomial-time Turing-reducibility is indeed a partial order relation (i.e., it is transitive and reflexive).

A language L is NP-*hard* (technically, NP-hard under Turing reductions[6]) if $\mathsf{NP} \subseteq \mathsf{P}^L$. Informally, NP-hard means "at least as hard as any NP problem": if we had a black box for an NP-hard problem, we could use it to solve all NP problems in polynomial time. Also, L is NP-*complete* if L is NP-hard *and* $L \in \mathsf{NP}$. Informally, NP-complete problems are the hardest problems in NP. (See Fig. 1.)

A priori, it is not completely obvious that NP-hard or NP-complete problems even exist. The great discovery of theoretical computer science in the 1970s was that hundreds of problems of practical importance fall into these classes: indeed, what is unusual is to find a hard NP problem that is *not* NP-complete.

More concretely, consider the following languages:

- 3SAT is the language consisting of all encodings of Boolean formulas φ over n variables, which consist of ANDs of "3-clauses" (i.e., ORs of up to three variables or their negations), such that there exists at least one assignment that satisfies φ. Here is an example, for which one can check that there's *no* satisfying assignment:

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

[6]In practice, often one only needs a special kind of Turing reduction called a *many-one reduction* or *Karp reduction*, which is a polynomial-time algorithm that maps every yes-instance of L' to a yes-instance of L, and every no-instance of L' to a no-instance of L. The additional power of Turing reductions—to make multiple queries to the L-oracle (with later queries depending on the outcomes of earlier ones), post-process the results of those queries, etc.—is needed only in a minority of cases. Nevertheless, for conceptual simplicity, throughout this survey I'll talk in terms of Turing reductions.

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

All of these languages are easily seen to be in NP. The famous *Cook-Levin Theorem* says that one of them—3Sat—is also NP-hard, and hence NP-complete.

Theorem 2 (Cook-Levin Theorem [67, 141]). 3Sat *is* NP*-complete.*

A proof of Theorem 2 can be found in any theory of computing textbook (for example, [208]). Here I'll confine myself to saying that Theorem 2 can be proved in three steps, each of them routine from today's standpoint:

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

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

where $\langle M\rangle$ is a description of the Turing machine M.

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

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

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

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

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

Once one knows that 3SAT is NP-complete, "the floodgates are open." One can then prove that countless other NP problems are NP-complete by reducing 3SAT to them, and then reducing those problems to others, and so on. The first indication of how pervasiveness NP-completeness really was came from Karp [122] in 1972. He showed, among many other results:

Theorem 3 (Karp [122]). HAMILTONCYCLE, TSP, CLIQUE, SUBSETSUM, *and* 3COL *are all* NP*-complete.*

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

Note that, if any NP-complete problem is in P, then all of them are, and P = NP. Conversely, if any NP-complete problem is not in P, then none of them are, and P ≠ NP.

One application of NP-completeness is to reduce the number of logical quantifiers needed to state the P ≠ NP conjecture. Let $\mathcal{PT}$ be the set of all polynomial-time Turing machines, and given a language L, let $L(x) = 1$ if $x \in L$ and $L(x) = 0$ otherwise. Then a "naïve" statement of P ≠ NP would be

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

(Here, by quantifying over all languages in NP, we really mean quantifying over all verification algorithms that define such languages.) Once we know that 3SAT (for example) is NP-complete, we can state P ≠ NP as simply:

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

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

2.2 Other Core Concepts

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

2.2.1 Search, Decision, and Optimization

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

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

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

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

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

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

  Otherwise, next ask:

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

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

Note that there *are* problems for which finding a solution is believed to be much harder than deciding whether one exists. A classic example, as it happens, is the problem of finding a Nash equilibrium of a matrix game. Here Nash's theorem guarantees that an equilibrium always exists, but an important 2006 result of Daskalakis et al. [71] gave evidence that there is no polynomial-time algorithm to *find* an equilibrium.[7] The upshot of Proposition 4 is just that search and decision are equivalent for the NP-*complete* problems.

[7]Technically, Daskalakis et al. showed that the search problem of finding a Nash equilibrium is complete for a complexity class called PPAD. This could be loosely interpreted as saying that the