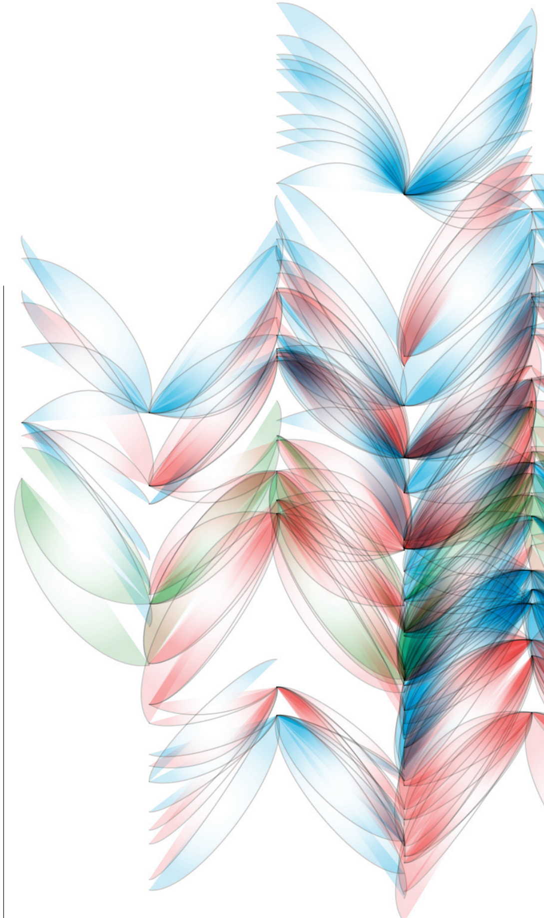
DOI:10.1145/1562164.1562186

It's one of the fundamental mathematical problems of our time, and its importance grows with the rise of powerful computers.

BY LANCE FORTNOW

The Status of the P versus NP Problem



The software written for this illustration makes a stylized version of a network graph that draws connections between elements based on proximity. The graph constantly changes as the elements sort themselves.

WHEN EDITOR-IN-CHIEF MOSHE Vardi asked me to write this piece for *Communications*, my first reaction was the article could be written in two words:

Still open.

When I started graduate school in the mid-1980s, many believed that the quickly developing area of circuit complexity would soon settle the **P** versus **NP** problem, whether every algorithmic problem with efficiently verifiable solutions have efficiently computable solutions. But circuit complexity and other approaches to the problem have stalled and we have little reason to believe we will see a proof separating **P** from **NP** in the near future.

Nevertheless, the computer science landscape has dramatically changed in the nearly four decades since Steve Cook presented his seminal **NP**-completeness paper "The Complexity of Theorem-Proving Procedures"[10] in Shaker Heights, OH in early May, 1971. Computational power has dramatically increased, the cost of computing has dramatically decreased, not to mention the power of the Internet. Computation has become a standard tool in just about every academic field. Whole subfields of biology, chemistry, physics, economics and others are devoted to large-scale computational modeling, simulations, and problem solving.

As we solve larger and more complex problems with greater computational power and cleverer algorithms, the problems we cannot tackle begin to stand out. The theory of **NP**-completeness helps us understand these limitations and the **P** versus **NP** problem begins to loom large not just as an interesting theoretical question in computer science, but as a basic principle that permeates all the sciences.

So while we don't expect the **P** versus **NP** problem to be resolved in the near future, the question has driven research in a number of topics to help us understand, handle, and even take

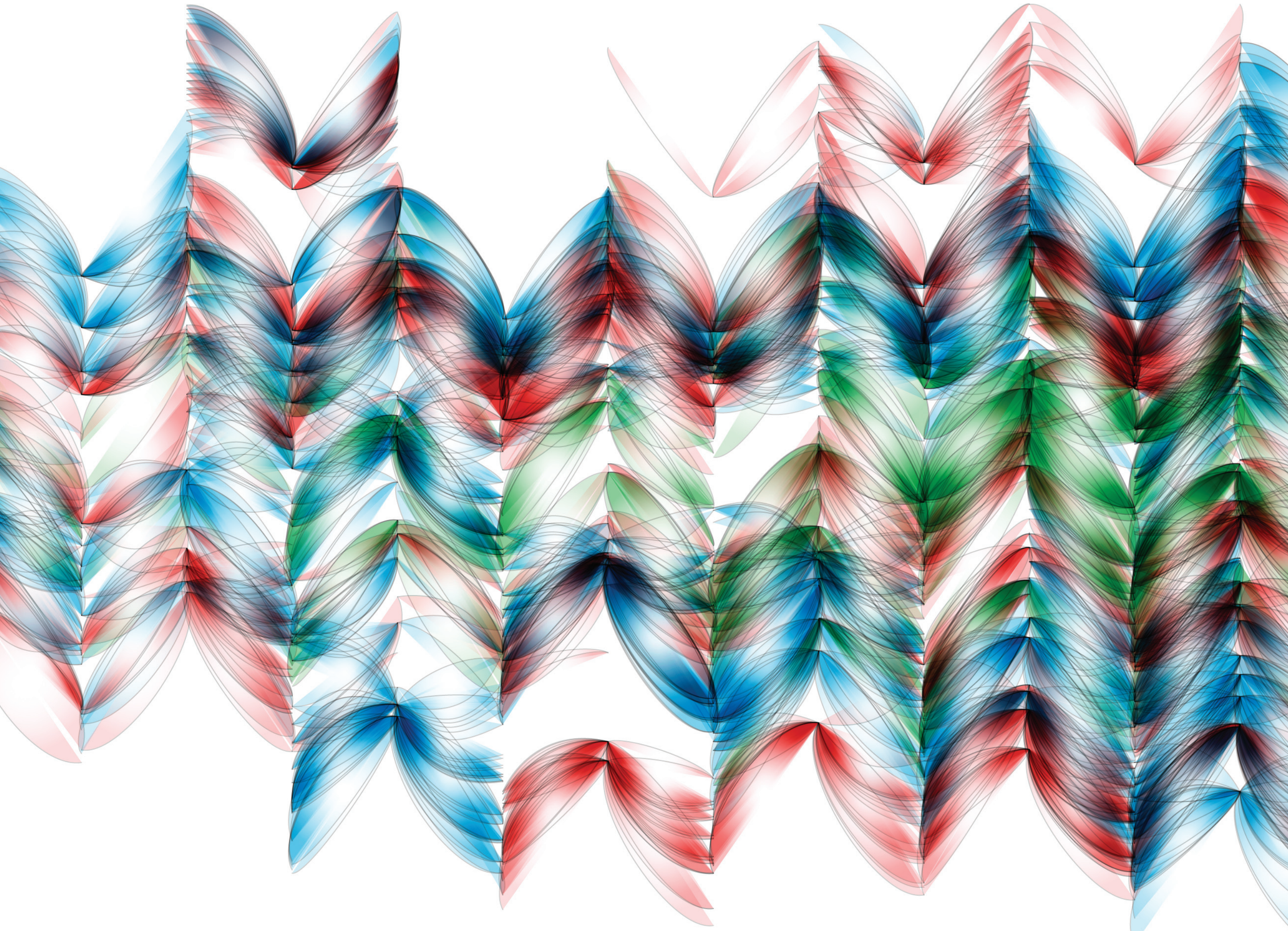


ILLUSTRATION BY C.E.B. REAS

advantage of the hardness of various computational problems.

In this article I look at how people have tried to solve the **P** versus **NP** problem as well as how this question has shaped so much of the research in computer science and beyond. I will look at how to handle **NP**-complete problems and the theory that has developed from those approaches. I show how a new type of "interactive proof systems" led to limitations of approximation algorithms and consider whether quantum computing can solve **NP**-complete problems (short answer: not likely). And I close by describing a new long-term project that will try to separate **P** from **NP** using algebraic-geometric techniques.

This article does not try to be totally accurate or complete either technically or historically, but rather informally describes the **P** versus **NP** problem and the major directions in computer science inspired by this question over the past several decades.

What is the P versus NP Problem?

Suppose we have a large group of students that we need to pair up to work on projects. We know which students are compatible with each other and we want to put them in compatible groups of two. We could search all possible pairings but even for 40 students we would have more than 300 billion trillion possible pairings.

In 1965, Jack Edmonds[12] gave an efficient algorithm to solve this matching problem and suggested a formal definition of "efficient computation" (runs in time a fixed polynomial of the input size). The class of problems with efficient solutions would later become known as **P** for "Polynomial Time."

But many related problems do not seem to have such an efficient algorithm. What if we wanted to make groups of three students with each pair of students in each group compatible (Partition into Triangles)? What if we wanted to find a large group of students all of whom are compatible with each other (Clique)? What if we wanted to sit students around a large round table with no incompatible students sitting next to each other (Hamiltonian Cycle)? What if we put the students into three groups so that each student is in the same group with only his or her compatibles (3-Coloring)?

All these problems have a similar favor: Given a potential solution, for example, a seating chart for the round table, we can validate that solution efficiently. The collection of problems that have efficiently verifiable solutions is known as **NP** (for "Nondeterministic Polynomial-Time," if you have to ask).

So **P** = **NP** means that for every problem that has an efficiently verifiable solution, we can find that solution efficiently as well.

We call the very hardest **NP** problems (which include Partition into Triangles, Clique, Hamiltonian Cycle and 3-Coloring) "NP-complete," that is, given an efficient algorithm for one of them, we can find an efficient algorithm for all of them and in fact any problem in **NP**. Steve Cook, Leonid Levin, and Richard Karp[10, 24, 27] developed the initial theory of **NP**-completeness that generated multiple ACM Turing Awards.

In the 1970s, theoretical computer scientists showed hundreds more problems **NP**-complete (see Garey and Johnson[16]). An efficient solution to any **NP**-complete problem would imply **P** = **NP** and an efficient solution to every **NP**-complete problem.

Most computer scientists quickly came to believe **P** ≠ **NP** and trying to prove it quickly became the single most important question in all of theoretical computer science and one of the most important in all of mathematics. Soon the **P** versus **NP** problem became an important computational issue in nearly every scientific discipline.

As computers grew cheaper and more powerful, computation started playing a major role in nearly every academic field, especially the sciences. The more scientists can do with computers, the more they realize some problems seem computationally difficult. Many of these fundamental problems turn out to be **NP**-complete. A small sample:

- Finding a DNA sequence that best fits a collection of fragments of the sequence (see Gusfield[20]).
- Finding a ground state in the Ising model of phase transitions (see Cipra[8]).
- Finding Nash Equilbriums with specific properties in a number of environments (see Conitzer[9]).
- Finding optimal protein threading procedures.[26]
- Determining if a mathematical statement has a short proof (follows from Cook[10]).

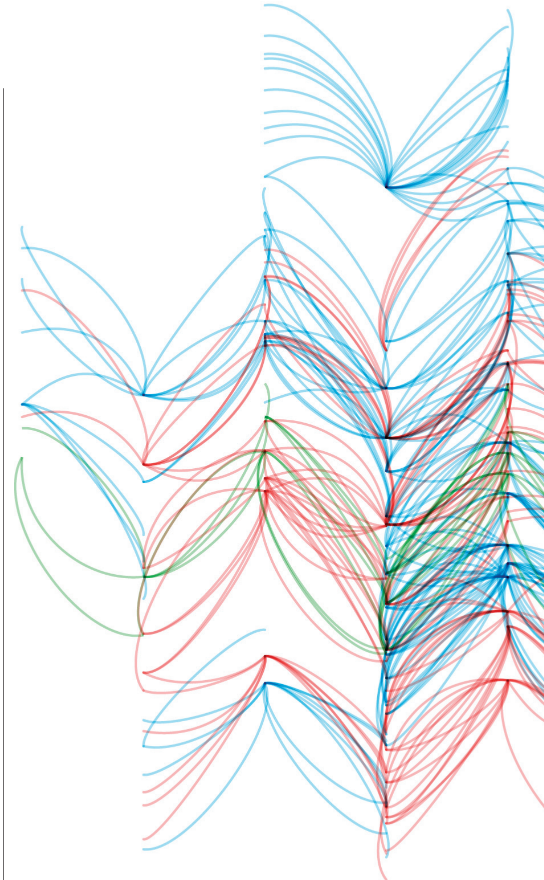
In 2000, the Clay Math Institute named the **P** versus **NP** problem as one of the seven most important open questions in mathematics and has offered a million-dollar prize for a proof that determines whether or not **P** = **NP**.

What If P = NP?

To understand the importance of the **P** versus **NP** problem let us imagine a world where **P** = **NP**. Technically we could have **P** = **NP**, but not have practical algorithms for most **NP**-complete problems. But suppose in fact we do have very quick algorithms for all these problems.

Many focus on the negative, that if **P** = **NP** then public-key cryptography becomes impossible. True, but what we will gain from **P** = **NP** will make the whole Internet look like a footnote in history.

What we would gain from P = NP will make the whole Internet look like a footnote in history.



Since all the **NP**-complete optimization problems become easy, everything will be much more efficient. Transportation of all forms will be scheduled optimally to move people and goods around quicker and cheaper. Manufacturers can improve their production to increase speed and create less waste. And I'm just scratching the surface.

Learning becomes easy by using the principle of Occam's razor—we simply find the smallest program consistent with the data. Near perfect vision recognition, language comprehension and translation and all other learning tasks become trivial. We will also have much better predictions of weather and earthquakes and other natural phenomenon.

P = **NP** would also have big implications in mathematics. One could find short, fully logical proofs for theorems

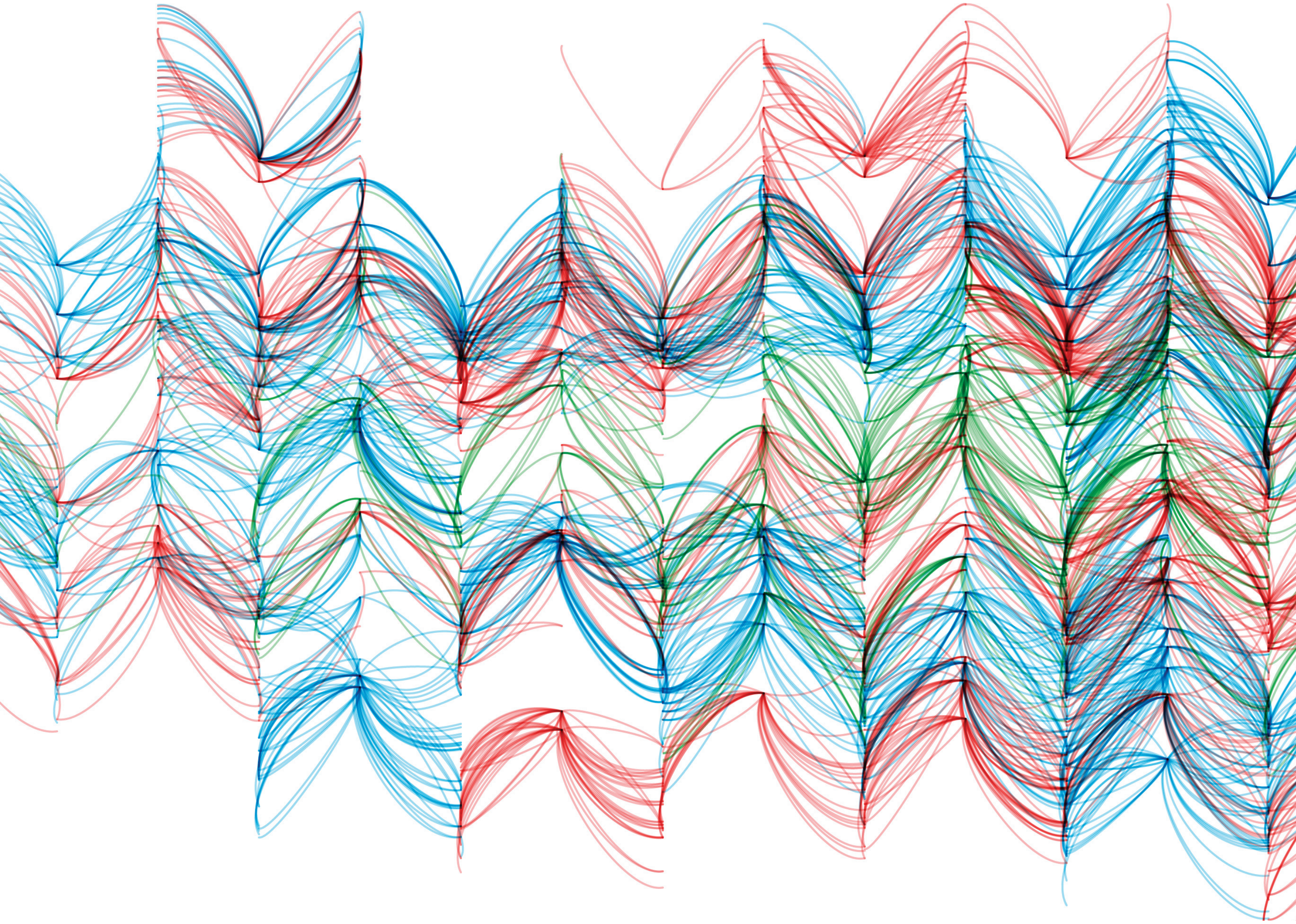


ILLUSTRATION BY C.E.B. REAS

but these proofs are usually extremely long. But we can use the Occam razor principle to recognize and verify mathematical proofs as typically written in journals. We can then find proofs of theorems that have reasonable length proofs say in under 100 pages. A person who proves **P** = **NP** would walk home from the Clay Institute not with $1 million check but with seven (actually six since the Poincaré Conjecture appears solved).

Don't get your hopes up. Complexity theorists generally believe **P** ≠ **NP** and such a beautiful world cannot exist.

Approaches to Showing P ≠ NP

Here, I present a number of ways we have tried and failed to prove **P** ≠ **NP**. The survey of Fortnow and Homer[14] gives a fuller historical overview of these techniques.

Diagonalization. Can we just construct an **NP** language L specifically designed so that every single polynomial-time algorithm fails to compute L properly on some input? This approach, known as diagonalization, goes back to the 19th century.

In 1874, Georg Cantor[7] showed the real numbers are uncountable using a technique known as diagonalization. Given a countable list of reals, Cantor showed how to create a new real number not on that list.

Alan Turing, in his seminal paper on computation,[38] used a similar technique to show that the Halting problem is not computable. In the 1960s complexity theorists used diagonalization to show that given more time or memory one can solve more problems. Why not use diagonalization to separate **NP** from **P**?

Diagonalization requires simulation and we don't know how a fixed **NP** machine can simulate an arbitrary **P** machine. Also a diagonalization proof would likely relativize, that is, work even if all machines involved have access to the same additional information. Baker, Gill and Solovay[6] showed no relativizable proof can settle the **P** versus **NP** problem in either direction.

Complexity theorists have used diagonalization techniques to show some **NP**-complete problems like Boolean formula satisfiability cannot have algorithms that use both a small amount of time and memory,[39] but this is a long way from **P** ≠ **NP**.

Circuit Complexity. To show **P** ≠ **NP** it is sufficient to show some -complete problem cannot be solved by relatively small circuits of AND, OR, and NOT gates (the number of gates bounded by a fixed polynomial in the input size).

In 1984, Furst, Saxe, and Sipser[15] showed that small circuits cannot solve the parity function if the circuits have a fixed number of layers of gates. In 1985, Razborov[31] showed the **NP**-complete problem of finding a large clique does not have small circuits if one only allows AND and OR gates (no NOT gates). If one extends Razborov's result to general circuits one will have proved **P** ≠ **NP**.

Razborov later showed his techniques would fail miserably if one allows NOT gates.[32] Razborov and Rudich[33] develop a notion of "natural" proofs and give evidence that our limited techniques in circuit complexity cannot be pushed much further. And, in fact, we haven't seen any significantly new circuit lower bounds in the past 20 years.

Proof Complexity. Consider the set of Tautologies, the Boolean formulas ϕ of variables over ANDs, ORs, and NOTs such that every setting of the variables to True and False makes ϕ true, for example the formula

$$(x \text{ AND } y) \text{ OR } (\text{NOT } x) \text{ OR } (\text{NOT } y).$$

A literal is a variable or its negation, such as x or NOT x. A formula, like the one here, is in Disjunctive Normal Form (DNF) if it is the OR of ANDs of one or more literals.

If a formula ϕ is not a tautology, we can give an easy proof of that fact by exhibiting an assignment of the variables that makes ϕ false. But if ϕ were indeed a tautology, we don't expect short proofs. If one could prove there are no short proofs of tautology that would imply **P** ≠ **NP**.

Resolution is a standard approach to proving tautologies of DNFs by finding two clauses of the form (ψ_1 AND x) and (ψ_2 AND NOT x) and adding the clause (ψ_1 AND ψ_2). A formula is a tautology exactly when one can produce an empty clause in this manner.

In 1985, Wolfgang Haken[21] showed that tautologies that encode the pigeonhole principle ($n + 1$ pigeons in n holes means some hole has more than one pigeon) do not have short resolution proofs.

Since then complexity theorists have shown similar weaknesses in a number of other proof systems including cutting planes, algebraic proof systems based on polynomials, and restricted versions of proofs using the Frege axioms, the basic axioms one learns in an introductory logic course.

But to prove **P** ≠ **NP** we would need to show that tautologies cannot have short proofs in an arbitrary proof system. Even a breakthrough result showing tautologies don't have short general Frege proofs would not suffice in separating **NP** from **P**.

Dealing with Hardness

So you have an **NP**-complete problem you just have to solve. If, as we believe, **P** ≠ **NP** you won't find a general algorithm that will correctly and accurately solve your problem all the time. But sometimes you need to solve the problem anyway. All hope is not lost. Here, I describe some of the tools one can use on **NP**-complete problems and how computational complexity theory studies these approaches. Typically one needs to combine several of these approaches when tackling **NP**-complete problems in the real world.

Brute Force. Computers have gotten faster, much faster since **NP**-completeness was first developed. Brute force search through all possibilities is now possible for some small problem instances. With some clever algorithms we can even solve some moderate size problems with ease.

The **NP**-complete traveling salesperson problem asks for the smallest distance tour through a set of specified cities. Using extensions of the cutting-plane method we can now solve, in practice, traveling salespeople problems with more than 10,000 cities (see Applegate[3]).

Consider the 3SAT problem, solving Boolean formula satisfiability where formulas are in the form of the AND of several clauses where each clause is the OR of three literal variables or negations of variables). 3SAT remains **NP**-complete but the best algorithms can in practice solve SAT problems on about 100 variables. We have similar results for other variations of satisfiability and many other **NP**-complete problems.

But for satisfiability on general formulae and on many other **NP**-complete problems we do not know algorithms better than essentially searching all the possibilities. In addition, all these algorithms have exponential growth in their running times, so even a small increase in the problem size can kill what was an efficient algorithm. Brute force alone will not solve **NP**-complete problems no matter how clever we are.

Parameterized Complexity. Consider the Vertex Cover problem, find a set of k "central people" such that for every compatible pair of people, at least one of them is central. For small k we can determine whether a central set of people exists efficiently no matter the total number n of people we are considering. For the Clique problem even for small k the problem can still be difficult.

Downey and Fellows[11] developed a theory of parameterized complexity that gives a fine-grained analysis of the complexity of **NP**-complete problems based on their parameter size.

Approximation. We cannot hope to solve **NP**-complete optimization problems exactly but often we can get a good approximate answer. Consider the traveling salesperson problem again with distances between cities given as the crow flies (Euclidean distance). This problem remains **NP**-complete but Arora[4] gives an efficient algorithm that gets very close to the best possible route.

Consider the MAX-CUT problem of dividing people into two groups to maximize the number of incompatibles between the groups. Goemans and Williamson[17] use semi-definite programming to give a division of people only a .878567 factor of the best possible.

Heuristics and Average-Case Complexity. The study of **NP**-completeness focuses on how algorithms perform on the worst possible inputs. However the specific problems that arise in practice may be much easier to solve. Many computer scientists employ various heuristics to solve **NP**-complete problems that arise from the specific problems in their fields.

While we create heuristics for many of the **NP**-complete problems, Boolean formula Satisfiability (SAT) receives more attention than any other. Boolean formulas, especially those in conjunctive normal form (CNF), the AND of ORs of variables and their negations, have a very simple description and yet are general enough to apply to a large number of practical scenarios particularly in software verification and artificial intelligence. Most natural **NP**-complete problems have simple efficient reductions to the satisfiability of Boolean formulas. In competition these SAT solvers can often settle satisfiability of formulas of one million variables.[a]

Computational complexity theorists study heuristics by considering average-case complexity—how well can algorithms perform on average from instances generated by some specific distribution.

Leonid Levin[28] developed a theory of efficient algorithms over a specific distribution and formulated a distributional version of the **P** versus **NP** problem.

Some problems, like versions of the

a http://www.satcompetition.org.

shortest vector problem in a lattice or computing the permanent of a matrix, are hard on average exactly when they are hard on worst-case inputs, but neither of these problems is believed to be **NP**-complete. Whether similar worst-to-average reductions hold for **NP**-complete sets is an important open problem.

Average-case complexity plays an important role in many areas of computer science, particularly cryptography, as discussed later.

Interactive Proofs and Limits of Approximation

Previously, we saw how sometimes one can get good approximate solutions to **NP**-complete optimization problems. Many times though we seem to hit a limit on our ability to even get good approximations. We now know that we cannot achieve better approximations on many of these problems unless **P** = **NP** and we could solve these problems exactly. The techniques to show these negative results came out of a new model of proof system originally developed for cryptography and to classify group theoretic algorithmic problems.

As mentioned earlier, we don't expect to have short traditional proofs of tautologies. But consider an "interactive proof" model where a prover Peggy tries to convince a verifier Victor that a formula ϕ is a tautology. Victor can ask Peggy randomly generated questions and need only be convinced with high confidence. Quite surprisingly, these proof systems have been shown to exist not only for tautologies but for any problem computable in a reasonable amount of memory.

A variation known as a "probabilistically checkable proof system" (PCPs), where Peggy writes down an encoded proof and Victor can make randomized queries to the bits of the proof, has applications for approximations. The "PCP Theorem" optimizes parameters, which in its strong form shows that every language in **NP** has a PCP where Victor uses a tiny number of random coins and queries only three bits of the proof.

One can use this PCP theorem to show the limitations of approximation for a large number of optimization questions. For example, one cannot approximate the largest clique in a group of n people by more than a multiplicative ratio of nearly $\sqrt{n}$ unless **P** = **NP**. See Madhu Sudan's recent article in *Communications* for more details and references on PCPs.[36]

We expect P ≠ NP to hold in very strong ways. We can use strong hardness assumptions as a positive tool, particularly to create cryptographic protocols and to reduce or even eliminate the need of random bits in probabilistic algorithms.

One can do even better assuming a "Unique Games Conjecture" that there exists PCPs for **NP** problems with some stronger properties. Consider the MAX-CUT problem of dividing people discussed earlier. If the unique games conjecture holds one cannot do better than the .878567 factor given by the Goemans-Williamson approximation algorithm.[26] Recent work shows how to get a provably best approximation for essentially any constrained problem assuming this conjecture.[30]

Using Hardness

In "What If **P** = **NP**?" we saw the nice world that arises when we assume **P** = **NP**. But we expect **P** ≠ **NP** to hold in very strong ways. We can use strong hardness assumptions as a positive tool, particularly to create cryptographic protocols and to reduce or even eliminate the need of random bits in probabilistic algorithms.

Cryptography. We take it for granted these days, the little key or lock on our Web page that tells us that someone listening to the network won't get the credit card number I just sent to an online store or the password to the bank that controls my money. But public-key cryptography, the ability to send secure messages between two parties that have never privately exchanged keys, is a relatively new development based on hardness assumptions of computational problems.

If **P** = **NP** then public-key cryptography is impossible. Assuming **P** ≠ **NP** is not enough to get public-key protocols, instead we need strong average-case assumptions about the difficulty of factoring or related problems.

We can do much more than just public-key cryptography using hard problems. Suppose Alice's husband Bob is working on a Sudoku puzzle and Alice claims she has a solution to the puzzle (solving a $n \times n$ Sudoku puzzle is **NP**-complete). Can Alice convince Bob that she knows a solution without revealing any piece of it?

Alice can use a "zero-knowledge proof," an interactive proof with the additional feature that the verifier learns nothing other than some property holds, like a Sudoku puzzle having a solution. Every **NP** search problem has a zero-knowledge proof under the appropriate hardness assumptions.

Online poker is generally played through some "trusted" Web site, usually somewhere in the Caribbean. Can we play poker over the Internet without a trusted server? Using the right cryptographic assumptions, not only poker but any protocol that uses a trusted party can be replaced by one that uses no trusted party and the players can't cheat or learn anything new beyond what they could do with the trusted party.[b]

Eliminating Randomness. In the 1970s we saw a new type of algorithm, one that used random bits to aid in finding a solution to a problem. Most notably we had probabilistic algorithms[35] for determining whether a number is prime, an important routine needed for modern cryptography. In 2004, we discovered we don't need randomness at all to efficiently determine if a number is prime.[2] Does randomness help us at all in finding solutions to **NP** problems?

Truly independent and uniform random bits are either very difficult or impossible to produce (depending on your beliefs about quantum mechanics). Computer algorithms instead use pseudorandom generators to generate a sequence of bits from some given seed. The generators typically found on our computers usually work well but occasionally give incorrect results both in theory and in practice.

We can create theoretically better pseudorandom generators in two different ways, one based on the strong hardness assumptions of cryptography and the other based on worst-case complexity assumptions. I will focus on this second approach.

We need to assume a bit more than **P** ≠ **NP**, roughly that **NP**-complete problems cannot be solved by smaller than expected AND-OR-NOT circuits. A long series of papers showed that, under this assumption, any problem with an efficient probabilistic algorithm also has an efficient algorithm that uses a pseudorandom generator with a very short seed, a surprising connection between hard languages and pseudo-randomness (see Impagliazzo[23]). The seed is so short we can try all possible seeds efficiently and avoid the need for randomness altogether.

Thus complexity theorists generally believe having randomness does not help in solving **NP** search problems and that **NP**-complete problems do not have efficient solutions, either with or without using truly random bits.

While randomness doesn't seem necessary for solving search problems, the unpredictability of random bits plays a critical role in cryptography and interactive proof systems and likely cannot be avoided in these scenarios.

Could Quantum Computers Solve NP-Complete Problems?

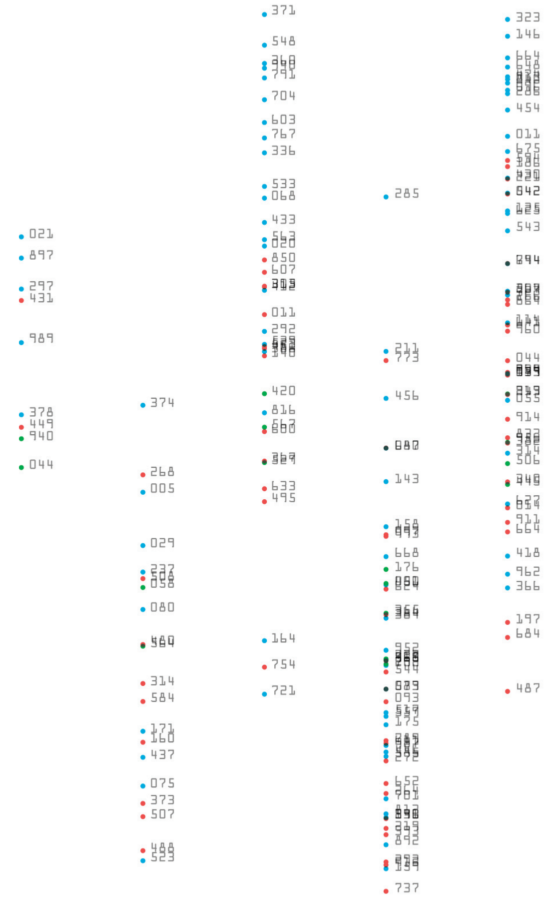
While we have randomized and nonrandomized efficient algorithms for determining whether a number is prime, these algorithms usually don't give us the factors of a composite number. Much of modern cryptography relies on the fact that factoring or similar problems do not have efficient algorithms.

In the mid-1990s, Peter Shor[34] showed how to factor numbers using a hypothetical quantum computer. He also developed a similar quantum algorithm to solve the discrete logarithm problem. The hardness of discrete logarithm on classical computers is also used as a basis for many cryptographic protocols. Nevertheless, we don't expect that factoring or finding discrete logarithms are **NP**-complete. While we don't think we have efficient algorithms to solve factoring or discrete logarithm, we also don't believe we can reduce **NP**-complete problems like Clique to the factoring or discrete logarithm problems.

So could quantum computers one day solve **NP**-complete problems? Unlikely.

I'm not a physicist so I won't address the problem as to whether these machines can actually be built at a large enough scale to solve factoring problems larger than we can with current technology (about 200 digits). After billions of dollars of funding of quantum computing research we still have a long way to go.

Even if we could build these machines, Shor's algorithm relies heavily on the algebraic structures of numbers that we don't see in the known **NP**-complete problems. We know that his algorithm cannot be applied to generic "black-box" search problems so any algorithm would have to use some special structure of **NP**-complete problems that we don't know about. We have used some algebraic structure of **NP**-complete problems for interactive and zero-knowledge proofs but quantum algorithms would seem to require much more.



NP can be seen as a graph where every element is connected to every other element. Over these pages a deconstruction of the graph is shown.

Lov Grover[19] did find a quantum algorithm that works on general **NP** problems but that algorithm only achieves a quadratic speed-up and we have evidence that those techniques will not go further.

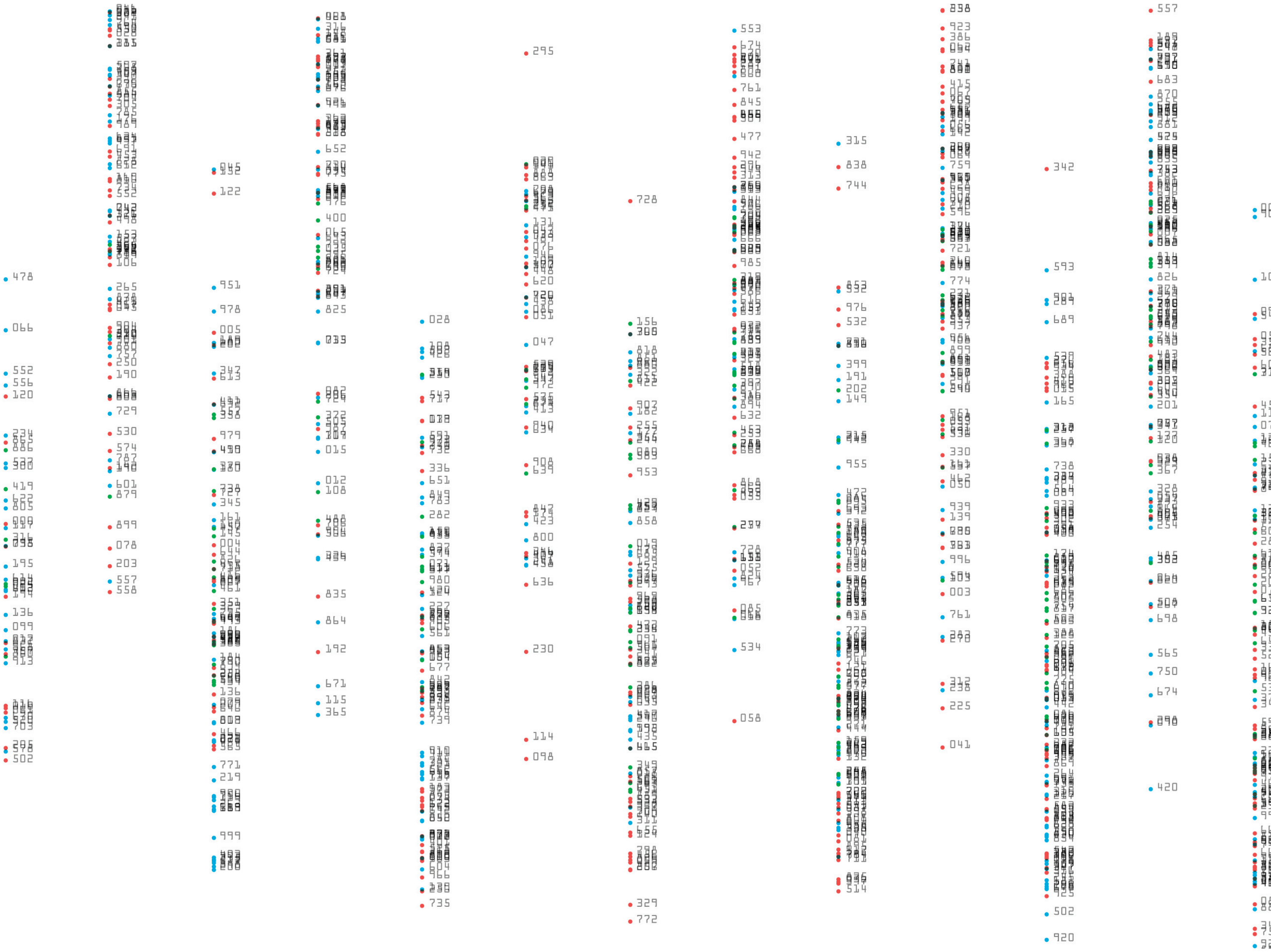
Meanwhile quantum cryptography, using quantum mechanics to achieve some cryptographic protocols without hardness assumptions, has had some success both in theory and in practice.

A New Hope?

Ketan Mulmuley and Milind Sohoni have presented an approach to the **P** versus **NP** problem through algebraic geometry, dubbed Geometric Complexity Theory, or GCT.[29]

This approach seems to avoid the dif-

b See the survey of Goldreich[18] for details.



ficulties mentioned earlier, but requires deep mathematics that could require many years or decades to carry through.

In essence, they define a family of high-dimension polygons P_n based on group representations on certain algebraic varieties. Roughly speaking, for each n, if P_n contains an integral point, then any circuit family for the Hamiltonian path problem must have size at least $n^{\log n}$ on inputs of size n, which implies **P** ≠ **NP**. Thus, to show that **P** ≠ **NP** it suffices to show that P_n contains an integral point for all n.

Although all that is necessary is to show that P_n contains an integral point for all n, Mulmuley and Sohoni argue that this direct approach would be difficult and instead suggest first showing that the integer programming problem for the family P_n is, in fact, in **P**. Under this approach, there are three significant steps remaining:

1. Prove that the LP relaxation solves the integer programming problem for P_n in polynomial time;
2. Find an efficient, simple combinatorial algorithm for the integer programming problem for P_n, and;
3. Prove that this simple algorithm always answers "yes."

Since the polygons P_n are algebro-geometric in nature, solving (1) is thought to require algebraic geometry, representation theory, and the theory of quantum groups. Mulmuley and Sohoni have given reasonable algebro-geometric conditions that imply (1). These conditions have classical analogues that are known to hold, based on the Riemann Hypothesis over finite fields (a theorem proved by André Weil in the 1960s). Mulmuley and Sohoni suggest that an analogous Riemann Hypothesis-like statement is required here (though not the classical Riemann Hypothesis).

Although step (1) is difficult, Mulmuley and Sohoni have provided definite conjectures based on reasonable mathematical analogies that would solve (1). In contrast, the path to completing steps (2) and (3) is less clear. Despite these remaining hurdles, even solving the conjectures involved in (1) could provide some insight to the **P** versus **NP** problem.

Mulmuley and Sohoni have reduced a question about the nonexistence of polynomial-time algorithms for all **NP**-complete problems to a question about the existence of a polynomial-time algorithm (with certain properties) for a specific problem. This should give us some hope, even in the face of problems (1)–(3).

Nevertheless, Mulmuley believes it will take about 100 years to carry out this program, if it works at all.

Conclusion

This survey focused on the **P** versus **NP** problem, its importance, our attempts to prove **P** ≠ **NP** and the approaches we use to deal with the **NP**-complete problems that nature and society throws at us. Much of the work mentioned required a long series of mathematically difficult research papers that I could not hope to adequately cover in this short article. Also the field of computational complexity goes well

ILLUSTRATION BY C.E.B. REAS

beyond just the **P** versus **NP** problem that I haven't discussed here. In "Further Reading," a number of references are presented for those interested in a deeper understanding of the **P** versus **NP** problem and computational complexity.

The **P** versus **NP** problem has gone from an interesting problem related to logic to perhaps the most fundamental and important mathematical question of our time, whose importance only grows as computers become more powerful and widespread. The question has even hit popular culture appearing in television shows such as *The Simpsons* and *Numb3rs*. Yet many only know of the basic principles of **P** versus **NP** and I hope this survey has given you a small feeling of the depth of research inspired by this mathematical problem.

Proving **P** ≠ **NP** would not be the end of the story, it would just show that **NP**-complete problem, don't have efficient algorithms for all inputs but many questions might remain. Cryptography, for example, would require that a problem like factoring (not believed to be **NP**-complete) is hard for randomly drawn composite numbers.

Proving **P** ≠ **NP** might not be the start of the story either. Weaker separations remain perplexingly difficult, for example showing that Boolean-formula Satisfiability cannot be solved in near-linear time or showing that some problem using a certain amount of memory cannot be solved using roughly the same amount of time.

None of us truly understands the **P** versus **NP** problem, we have only begun to peel the layers around this increasingly complex question. Perhaps we will see a resolution of the **P** versus **NP** problem in the near future but I almost hope not. The **P** versus **NP** problem continues to inspire and boggle the mind and continued exploration of this problem will lead us to yet even new complexities in that truly mysterious process we call computation.

Further Reading

Recommendations for a more in-depth look at the **P** versus **NP** problem and the other topics discussed in this article:

- Steve Homer and I have written a detailed historical view of computational complexity.[14]
- The 1979 book of Garey and Johnson still gives the best overview of the **P** versus **NP** problem with an incredibly useful list of **NP**-complete problems.[16]
- Scott Aaronson looks at the unlikely possibility that the **P** versus **NP** problem is formally independent.[1]
- Russell Impagliazzo gives a wonderful description of five possible worlds of complexity.[22]
- Sanjeev Arora and Boaz Barak have a new computational complexity textbook with an emphasis on recent research directions.[5]
- The *Foundations and Trends in Theoretical Computer Science* journal and the Computational Complexity columns of the *Bulletin of the European Association of Theoretical Computer Scienc*e and *SIGACT News* have many wonderful surveys on various topics in theory including those mentioned in this article.
- Read the blog Computational Complexity and you will be among the first to know about any updates of the status of the **P** versus **NP** problem.[13]

Acknowledgments

Thanks to Rahul Santhanam for many useful discussions and comments. Josh Grochow wrote an early draft. The anonymous referees provided critical advice. Some of the material in this article has appeared in my earlier surveys and my blog.[13]

References

1. Aaronson, S. Is P versus NP formally independent? *Bulletin of the European Association for Theoretical Computer Science 81* (Oct. 2003).
2. Agrawal, M., Kayal, N., and Saxena, N. PRIMES. In *Annals of Mathematics 160*, 2 (2004) 781–793.
3. Applegate, D., Bixby, R., Chvátal, V., and Cook, W. On the solution of traveling salesman problems. *Documenta Mathematica*, Extra Volume ICM III (1998), 645–656.
4. Arora, S. Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. *J. ACM 45*, 5 (Sept. 1998), 753–782.
5. Arora, S. and Barak, B. *Complexity Theory: A Modern Approach.* Cambridge University Press, Cambridge, 2009.
6. Baker, T., Gill, J., and Solovay, R. Relativizations of the P = NP question. *SIAM Journal on Computing 4*, 4 (1975), 431–442.
7. Cantor, G. Ueber eine Eigenschaft des Inbegriffes aller reellen algebraischen Zahlen. *Crelle's Journal 77* (1874), 258–262.
8. Cipra, B. This Ising model is NP-complete. *SIAM News 33*, 6 (July/Aug. 2000).
9. Conitzer, V. and Sandholm, T. New complexity results about Nash equilibria. *Games and Economic Behavior 63*, 2 (July 2008), 621–641.
10. Cook, S. The complexity of theorem-proving procedures. In *Proceedings of the 3rd ACM Symposium on the Theory of Computing*, ACM, NY, 1971, 151–158.
11. Downey, R. and Fellows, M. *Parameterized Complexity.* Springer, 1999.
12. Edmonds, J. Paths, trees and owers. *Canadian Journal of Mathematics 17*, (1965), 449–467.
13. Fortnow, L. and Gasarch, W. Computational complexity; http://weblog.fortnow.com.
14. Fortnow, L. and Homer, S. A short history of computational complexity. *Bulletin of the European Association for Theoretical Computer Science 80*, (June 2003).
15. Furst, M., Saxe, J., and Sipser, M. Parity, circuits and the polynomial-time hierarchy. *Mathematical Systems Theory 17* (1984), 13–27.
16. Garey, M. and Johnson, D. *Computers and Intractability. A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, NY, 1979.
17. Goemans, M. and Williamson, D. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM 42*, 6 (1995), 1115–1145.
18. Goldreich, O. Foundations of cryptography|a primer. *Foundations and Trends in Theoretical Computer Science 1*, 1 (2005) 1–116.
19. Grover, L. A fast quantum mechanical algorithm for database search. In *Proceedings of the 28th ACM Symposium on the Theory of Computing.* ACM, NY, 1996, 212–219.
20. Gusfield, D. *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology.* Cambridge University Press, 1997.
21. Haken, A. The intractability of resolution. *Theoretical Computer Science, 39* (1985) 297–305.
22. Impagliazzo, R. A personal view of average-case complexity theory. In *Proceedings of the 10th Annual Conference on Structure in Complexity Theory*. IEEE Computer Society Press, 1995, 134–147.
23. Impagliazzo, R. and Wigderson, A. P = BPP if E requires exponential circuits: Derandomizing the XOR lemma. In *Proceedings of the 29th ACM Symposium on the Theory of Computing.* ACM, NY, 1997, 220–229.
24. Karp, R. Reducibility among combinatorial problems. *Complexity of Computer Computations.* R. Miller and J. Thatcher, Eds. Plenum Press, 1972, 85–103.
25. Khot, S., Kindler, G., Mossel, E., and O'Donnell, R. Optimal inapproximability results for MAX-CUT and other 2-variable CSPs? *SIAM Journal on Computing 37*, 1 (2007), 319–357.
26. Lathrop, R. The protein threading problem with sequence amino acid interaction preferences is NP-complete. *Protein Engineering 7*, 9 (1994), 1059–1068.
27. Levin, L. Universal'nyie perebornyie zadachi (Universal search problems: in Russian). *Problemy Peredachi Informatsii 9*, 3 (1973), 265–266. Corrected English translation.37
28. Levin, L. Average case complete problems. *SIAM Journal on Computing 15*, (1986), 285–286.
29. Mulmuley, K. and Sohoni, M. Geometric complexity theory I: An approach to the P vs. NP and related problems. *SIAM Journal on Computing 31*, 2, (2001) 496–526.
30. Raghavendra, P. Optimal algorithms and inapproximability results for every csp? In *Proceedings of the 40th ACM Symposium on the Theory of Computing.* ACM, NY, 2008, 245–254.
31. Razborov, A. Lower bounds on the monotone complexity of some Boolean functions. *Soviet Mathematics-Doklady 31*, (1985) 485–493.
32. Razborov, A. On the method of approximations. In *Proceedings of the 21st ACM Symposium on the Theory of Computing.* ACM, NY, 1989, 167–176.
33. Razborov, A., and Rudich, S. Natural proofs. *Journal of Computer and System Sciences 55*, 1 (Aug. 1997), 24–35.
34. Shor. P. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing 26*, 5 (1997) 1484–1509.
35. Solovay, R. and Strassen, V. A fast Monte-Carlo test for primality. *SIAM Journal on Computing 6* (1977), 84–85. See also erratum 7:118, 1978.
36. Sudan, M. Probabilistically checkable proofs. *Commun. ACM 52*, 3 (Mar. 2009) 76–84.
37. Trakhtenbrot, R. A survey of Russian approaches to Perebor (brute-force search) algorithms. *Annals of the History of Computing 6*, 4 (1984), 384–400.
38. Turing, A. On computable numbers, with an application to the Etscheidungs problem. *Proceedings of the London Mathematical Society 42* (1936), 230–265.
39. van Melkebeek, D. A survey of lower bounds for satisfiability and related problems. *Foundations and Trends in Theoretical Computer Science 2*, (2007), 197–303.

Lance Fortnow (fortnow@eecs.northwestern.edu) is a professor of electrical engineering and computer science at Northwestern University's McCormick School of Engineering, Evanston, IL.