The Limits of Quantum Computers (DRAFT)

Scott Aaronson

For the published version—which differs significantly from this one—please see the March 2008 issue of Scientific American.

"Haggar Physicists Develop 'Quantum Slacks,'" read a headline in the satirical weekly *The Onion*. By exploiting a bizarre "Schrödinger's Pants" duality, these non-Newtonian pants could paradoxically behave like formal wear and casual wear at the same time. The *Onion* writers were apparently spoofing the breathless accounts of quantum computing that have filled the popular press for years. If so, they picked an obvious target: many of those articles were ripe for parody. "Unlike a classical bit," the typical article might read, "a quantum bit—or 'qubit'—can be 0 and 1 at the same time. As a result, such a machine could instantly break 'unbreakable' codes by trying all possible keys at once."

It doesn't take an Einstein to smell something fishy in that assertion. After all, what would the world be like if there were a machine able to examine all possible solutions to a problem in parallel, and then immediately zero in on the right one like a falcon swooping down on its prey? To say that airlines could schedule their flights better, or that engineers could find ways to shoehorn more transistors onto a chip, is to miss the point entirely. For one thing, if such a machine existed, mathematicians would be permanently out of a job.

Why? Take your favorite conjecture that's remained unproved for centuries—like Goldbach's Conjecture, that every even number 4 or greater can be written a sum of two prime numbers. Now program the machine to search, not among *every* purported proof of that assertion, but "merely" among every proof with at most (say) a million symbols. The number of such proofs is finite, and each one—assuming it's written out in hairsplitting detail—can be rapidly checked by computer to see whether it's correct. So if quantum computers were really as powerful as the usual caricature suggests, then they could instantly find a correct proof, assuming there was one. And if there wasn't such a proof? Well then, try all proofs with ten million symbols, or a billion. Perhaps the logician Kurt Gödel said it best, in a 1956 letter to John von Neumann that first raised the possibility of such a hyperfast theorem-proving machine: "One would indeed have to simply select a [number of symbols] so large that, if the machine yields no result, there would then also be no reason to think further about the problem."

But if you had such a machine, the implications would go beyond pure mathematics. For example, you could ask a computer to search, among all possible parameter settings of a neural network, for the ones that do the best at predicting historical stock market data. Of course, there's no guarantee that those settings would predict the *future* prices better than the market, but it's a decent bet—as long as you were the only one who knew the settings! Or you could search for the shortest program that output the complete works of Shakespeare, after at most (say) 100 million operations. Again, it's possible that such a program would work in a boring way, by applying some souped-up text compression to the plays and sonnets. But if it were mathematically possible to create a shorter

program—by, say, by starting out from a simulation of Shakespeare's brain—then that shorter program is what you would find.

In short, it sounds like science fiction. So if a quantum computer would let us do all these things, then it's not surprising that, to hear it from some researchers, quantum computing *is* science fiction. Take, for example, the physicist and Nobel laureate Robert B. Laughlin, in his book *A Different Universe*: "I wish those who invest in quantum computing the best of luck. I also ask that anyone anxious to invest in a bridge in lower Manhattan contact me right away, for there will be discounts for a limited time only." Or the renowned computer scientist Leonid M. Levin: "The present attitude [of quantum computing researchers] is analogous to, say, Maxwell selling the Daemon of his famous thought experiment as a path to cheaper electricity from heat."

So what's going on? Will quantum computers let us transcend the human condition and become as powerful as gods? Or will the whole concept be exposed as a physical absurdity, as the twenty-first century's perpetual-motion machine? Over the past decade, while the early experimental steps toward building a quantum computer made the headlines, theoretical computer scientists were quietly working to figure out what a quantum computer, if we had one, would actually be able to compute. This research has revealed that even a quantum computer would face significant limitations.

Contrary to the popular image, their work has revealed even a quantum computer would face significant limitations. In particular, while a quantum computer could quickly factor large numbers, and thereby break most of the cryptographic codes used on the Internet today, there's reason to think that not even a quantum computer could solve the crucial class of NP-complete problems efficiently. This class includes the problems mentioned above. Limitations of quantum computers have also been found for games of strategy like chess, as well as for breaking cryptographic hash functions. All of these limitations apply to a class of algorithms known as "black-box algorithms," which encompasses all quantum algorithms known today. Giving a complete proof that *no* fast quantum algorithm can solve these problems is out of the question, since we can't even prove that no fast quantum algorithm can solve them.

Ironically, by revealing that quantum computers would face significant limitations, this understanding makes quantum computing less of an affront to intuition. This article is about the new understanding we've achieved, and what it means for the limits of efficient computation in the physical universe.

Firstly, we've learned, at least in principle, how to make quantum computers fault-tolerant—that is, resistant to unwanted interactions with the outside world, as long as the rate of those interactions is below a certain threshold (depending on the assumptions, perhaps one interaction per qubit per 100 clock cycles). The discovery of fault-tolerance put the ball firmly back in the skeptics' court, by answering what at the time was their central technical objection.

The consensus today is that, if quantum computing is shown to be impossible, this will be *much* more interesting from a fundamental physics standpoint than if it's shown to be possible. For it would mean that quantum mechanics, the bedrock theory of physics for the past 80 years, is wrong or incomplete.

But how could quantum computers offer such a dramatic speedup for one problem but not for another? And what do we mean by "speedup," anyway? Exactly how much faster are we talking? To even ask these questions correctly, let alone answer them, requires knowing something about one of great scientific ideas of the past fifty years: the theory of computational complexity. Unfortunately, even many scientists still don't understand this idea—but, after reading the next section, you will.

Complexity

Computational complexity theory deals with the resources needed to solve problems, in the *limit* as the problems get arbitrarily large. So it's not surprising that the theory emerged in the aftermath of World War II, the winning of which took the largest industrial effort the world had ever seen.

Let's start with P: the class of all problems solvable in polynomial time by a computer program. Some examples of problems in P are multiplication, division, searching mazes, sorting lists, and in general, almost all of the humdrum things that fill a computer's workday. I should point out that, for us computer scientists, it's the whole infinite *set* of mazes (or lists, etc.) that constitutes a "problem"—any particular maze is merely an "instance." The "size" of the instance is the amount of information needed to specify it: for example, the number of corridors in a maze or digits in a number. To be polynomial-time, a procedure has to solve *every* instance of size n using a number of steps that grows like n raised to a fixed power.

The Library of Babel, wrote Jorge Luis Borges, is comprised of "an indefinite and perhaps infinite number of hexagonal galleries," containing an untold number of copies of every possible book—or rather, every possible book of four hundred and ten pages, forty lines to a page, and eighty letters to a line. Most of the books consist entirely of gibberish. But somewhere in the Library, a patient enough searcher could find "everything: the minutely detailed history of the future, the archangels' autobiographies, the faithful catalogues of the Library, thousands and thousands of false catalogues, the demonstration of the fallacy of those catalogues, the Gnostic gospel of Basilides, the commentary on that gospel, the commentary on the commentary on that gospel…"

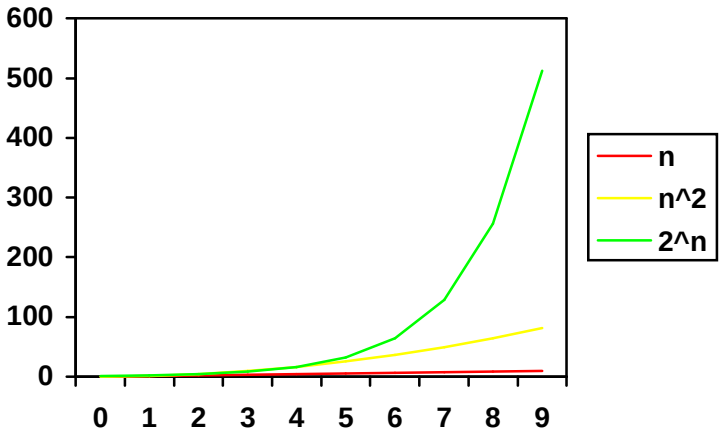
Borges was especially struck by the rarity of truth in the Library, as compared to meaninglessness and falsehood. Even if you stumbled on the secret of the universe in some forgotten tome, how could you ever distinguish it from the thousands of treacherous variations? But surely, some of the books would contain gems that a knowledgeable seeker would quickly recognize as such: proofs of Goldbach's conjecture, the Riemann hypothesis, and other statements that have eluded mathematicians for centuries; the

design for a controlled fusion reactor; a computer program that predicted the stock market better than any analyst; plays that Shakespeare could have written but didn't.

Part of what makes Borges' fantasy so compelling is that it *isn't* a fantasy: the Library of Babel exists. You enter it every time you start your word processor and face a blank screen and blinking cursor, the combinatorial cosmos laid out in front of you. But then why are so many of the books we'd most like to retrieve so inaccessible? What is it about our world that makes it easier to recognize a great novel than to write one, easier to be a critic than an artist?

One answer will be plunkingly obvious, at least to readers of this magazine: exponential growth. If there are 26 possibilities for the first letter of a book, then there are 676 possibilities for the first two letters, and 141,167,095,653,376 for the first ten. By the time we reach 100 letters, there are already more possibilities than there are atoms in a vat of 26^{100} atoms.

But if our goal were to find, say, a proof of the Riemann hypothesis at most a million symbols long, then we wouldn't need to examine literally *every* sequence of symbols. For example, if the beginning part of a sequence contained logical errors, or nonsense, or car ads, then we could eliminate every sequence with that same beginning from further consideration. But is a more dramatic improvement possible? In particular, could there be a procedure to find a proof with n symbols, using a number of steps that increased only *linearly* with n, or as n raised to a fixed power such as 2 or 3? Such a procedure would be called *polynomial-time*, since the time it used would be bounded by a polynomial function of the problem size [see figure below]. Computer scientists like to call a procedure "efficient" if it runs in polynomial time—a criterion that works surprisingly well in practice, though it would admittedly become strained if a procedure took n^{100} steps.



So, is there an efficient procedure to search the Library of Babel? If the very possibility strikes you as absurd, then consider some problems that might seem equally hopeless. Say you're given the names of a thousand men and women, along with a list of who's willing to marry whom, and want to pair everyone off with an agreeable spouse. Or—an example that arises in cryptography—say you're given a thousand-digit number, and you

want to decide whether it's prime or composite. In both cases, it's "obvious" that there's no way to avoid "brute-force" search, among all possible pairings in the one case and all possible divisors in the other. But in 1965, Jack Edmonds gave an ingenious method to pair everyone off whenever this is possible, in an amount of time that increases only polynomially with the number of people. (Interestingly, if we allow gay or lesbian pairings, then the problem becomes harder, but is still solvable in polynomial time.) A decade later, Robert Solovay and Volker Strassen, and independently Michael Rabin, found procedures to decide whether a number is prime or composite (although not to find the divisors of a composite number), using time that grows roughly as the number of digits cubed. The one defect of these procedures was that they produced the correct answer not with certainty, but only with overwhelming probability. However, in a dramatic advance three years ago, Manindra Agrawal, Neeraj Kayal, and Nitin Saxena gave a polynomial-time primality test that succeeds with certainty.

So, if we can eliminate "brute-force search" in the cases of matchmaking and primality testing, then why not for other problems, like proving theorems, breaking cryptographic codes, or writing a computer program to ace the S.A.T.? This, in essence, is the "P versus NP" question, whose place among the loftiest mysteries ever to try the human intellect has been confirmed by its cameo roles on at least two TV shows (*The Simpsons* and *NUMB3RS*). The question was first raised by the logician Kurt Gödel, of "This sentence is not provable" fame, in a remarkable 1956 letter to the computer pioneer John von Neumann. Gödel wanted to know whether there was a general method to find proofs of size n, using time that increased only as n or n^2. If such a method existed, Gödel argued that this

> "would have consequences of the greatest magnitude. That is to say, it would clearly indicate that the mental effort of the mathematician in the case of yes-or-no questions could be completely 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."

Von Neumann was dying of cancer at the time, and apparently never answered Gödel's letter. The question wouldn't become widely known until the discovery of "NP-completeness" in the 1970's, about which more later. Today, P versus NP heads the list of seven "Millennium Problems" for which the Clay Mathematics Institute is offering million-dollar prizes. One measure of its importance is that, depending on how you solved it, you could also program a computer to solve the other six problems.

But there's a further wrinkle. According to quantum mechanics, the bizarre tale that physicists have been telling since the 1920's, if you measure a beam of photons polarized at 45° angles to the measurement axis, the outcome will be a random sequence of horizontal and vertical polarizations: a random book from the Library of Babel written out in binary code. But until you measure the photons, you can't say that they encode *some* definite book, the only problem being that you don't know which one. For if you said that, then you'd make the wrong predictions for what would happen were you to measure the photons at a different angle. Instead, because of the quantum effect called interference, *all* possibilities can in general influence a measurement result. Whether that means that the possibilities literally constitute "parallel universes" is a question eagerly

debated by philosophers and stoners alike. But there's no getting around a basic fact: that in order to describe a state of ten million particles, our best-confirmed theory of the physical world requires us to specify an "amplitude" for *every one* of the $\sim 2^{10,000,000}$ possible outcomes of measuring the particles.

So you might wonder: if Nature is going to all that effort, then why not take advantage of it to perform enormous computations—perhaps even to search the Library of Babel in polynomial time? And indeed, in the 1980's, Richard Feynman, David Deutsch, and a few other physicists proposed building "quantum computers" that would outperform classical ones by an exponential amount. But supposing we did build such computers, how powerful would they actually be? For that matter, what about "relativity computers," or "superstring computers"? More generally, *what are the limits of efficient computation in Nature?*

This is at once a philosophical, scientific, and practical question, with implications ranging from the knowability of mathematical truth to the nature of the physical world to the future of the computer industry. Indeed, it's almost as audacious a question as anyone could ask—which makes it all the more surprising that computer scientists, physicists, and mathematicians have learned a great deal about it over the past fifteen years. But while the recent discoveries have been spectacular, they've also been difficult for outsiders to follow—phrased as they are in terms of exotic beasts called "complexity classes," with inscrutable names like BQP, NP, and PSPACE. Because of this, the understanding we've achieved has barely filtered into the popular press, and has remained all but absent even from books dealing with "complexity" (such as Stephen Wolfram's erroneously-titled *A New Kind of Science*).

This article is intended as a guide to the current frontier of computational complexity and physics. But before we get there, we need to revisit the question that started it all: P versus NP.

P versus NP

All right then, let's start with P: the class of all problems solvable in polynomial time by a computer program. Some examples of problems in P are multiplication, division, searching mazes, sorting lists, the matchmaking and primality problems mentioned earlier, and in general, almost all of the humdrum things that fill a computer's workday. I should point out that, for us computer scientists, it's the whole infinite *set* of mazes (or lists, etc.) that constitutes a "problem"—any particular maze is merely an "instance." The "size" of the instance is the amount of information needed to specify it: for example, the number of corridors in a maze or digits in a number. To be polynomial-time, a procedure has to solve *every* instance of size n using a number of steps that grows like n raised to a fixed power.

You might interject, "The number of steps in Java or C? Under Windows, Mac, or Linux?" The answer is that it doesn't matter. As Richard Feynman once said, computers today, like people today, are all fundamentally the same. Some are faster or have more

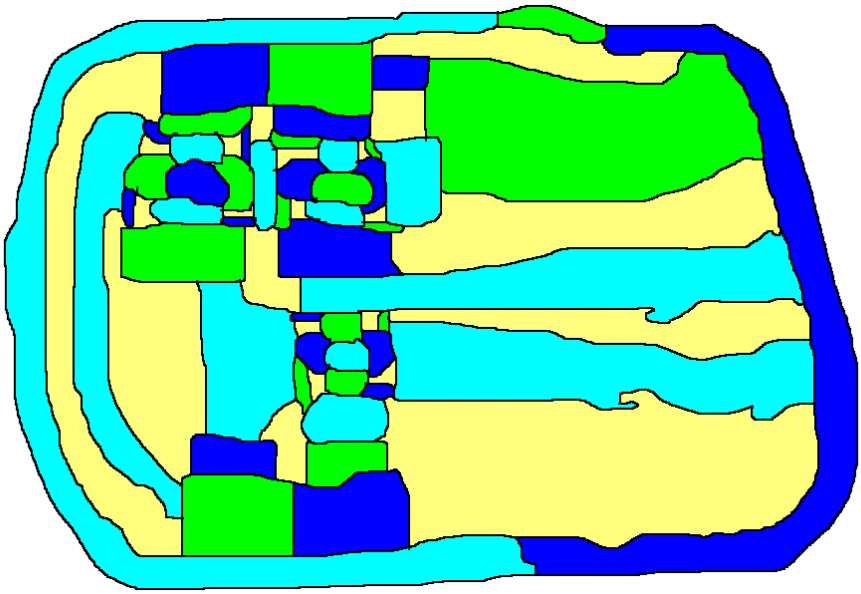
RAM than others, but they all hew to the rules laid down by Alan Turing in 1936. In particular, if you can solve a problem in polynomial time in Windows, then you can solve it in polynomial time on a Mac—for example, by just running the Windows program on an emulator. Indeed, the reason people are so excited about quantum computers is precisely that they seem violate this polynomial equivalence.

But what about NP? Contrary to widespread belief, NP does not stand for "Non Polynomial-Time," but for a technical concept called "Nondeterministic Polynomial-Time." (Yes, I know: when it comes to weird acronyms, computer scientists could put any Pentagon office to shame.) You can think of NP as the class of problems for which a valid solution can be *recognized* in polynomial time. For example, let's say you wanted to color a map with three colors, so that no two countries sharing a border were colored the same [see Figure]. With only a few countries, this is easy to do by trial and error, but as more countries are added, the number of possible colorings becomes astronomical. On the other hand, if you simply gave the coloring job to your grad student, then when the answer came back after the end of eternity, you could quickly tell whether or not your student had succeeded: you'd just have to look at the map, and check whether any two neighboring countries were colored the same. This is why the map-coloring problem is in NP.

It's clear that P is a subset of NP, since if you can solve a problem on your own, then you can certainly be persuaded that it *has* a solution. The question that keeps many of us up at night is whether P *equals* NP: in other words, can every NP problem be solved in polynomial time?

One thing we do know is that, instead of asking about *all* NP problems, we might as well ask about the map-coloring problem. Or the Maximum Clique problem (given which students in a high-school cafeteria will talk to whom, find the largest set of students who will all talk to each other). Or the problem of whether a given configuration can ever arise in the computer game Minesweeper. For it turns out that these problems, as well as thousands of others, *are all basically the same problem, and are all at least as hard as any other NP problem*. We know this because of the remarkable theory of "NP-completeness," which was created in the early 1970's by Stephen Cook and Richard Karp in the United States, and independently by Leonid Levin in the Soviet Union. A problem is NP-complete if it's among the "hardest" problems in NP. This means, first, that the problem is in NP, and second, that if a "black box" for solving the problem somehow washed up on the beach, then by using that black box, we could solve *any other* NP problem in polynomial time. In particular, if P=NP then every NP-complete problem is solvable in polynomial time, while if P≠NP then none of them are.

It wasn't clear *a priori* that any natural NP-complete problems even existed; this is what Cook and Levin showed. Today, if you want to prove your favorite problem is NP-complete, what you do is first find a problem that was already proved NP-complete, and then show that any instance of that problem can be efficiently "encoded as," or "transformed into," an instance of the problem you care about. For example, while you probably wouldn't guess by looking at it, the map in the following figure



actually encodes the Boolean expression

(x or y) and (x or not(y)) and (not(x) or y),

and from any valid coloring of the map you could read off the unique satisfying assignment, x=y=TRUE.

So why is it so hard to prove P≠NP? One answer is that any proof would need to distinguish the problems that *seem* hard but actually admit subtle algorithms (such as matchmaking and primality testing), from the "genuinely hard" NP-complete ones. A second answer comes from a 1975 theorem of Theodore Baker, John Gill, and Robert Solovay, who showed that "diagonalization"—the technique used by Gödel and Turing to prove their incompleteness and undecidability theorems—is not strong enough by itself to separate P from NP. A third answer has to do with the bizarre self-referential nature of P≠NP, a conjecture that all but asserts the difficulty of finding its own proof. In 1993, Steven Rudich and Alexander Razborov showed that most of the approaches then being tried on P≠NP and related conjectures would, if they worked, yield efficient algorithms to solve some of the very problems that they were supposed to prove intractable!

On the other hand, all it would take to show P=NP would be to solve *any* NP-complete problem in polynomial time. Since this hasn't happened yet, today it's almost universally believed that P≠NP, even if we're not yet smart enough to understand why. If we grant that assumption, then only one hope remains for ever combing the Library of Babel in polynomial time by computer: namely, to change what we mean by "computer."

Enter the Quantum Computer

The problem is that an ordinary computer is confined to operating on bits, which are always either 0 or 1, on or off. By contrast, a "quantum computer" could operate on quantum bits or 'qubits,' which can be 0 and 1 at the same time. It follows, of course,

that such a machine could examine all possible solutions to a problem at once, and instantly zero in on the correct one like a needle in a haystack.

Did that sound like a *non sequitur* to you? If it did, you're right. Over the past decade, we've learned that, even if experimentalists manage to surmount the staggering obstacles involved in building a quantum computer—and they might—we won't exactly transcend the human condition or become as powerful as gods. For it turns out that, while quantum computers would offer enormous speedups for a few problems, they'd also be subject to many of the same limitations as today's computers. In particular, there's reason to believe that not even a quantum computer could solve NP-complete problems in polynomial time.

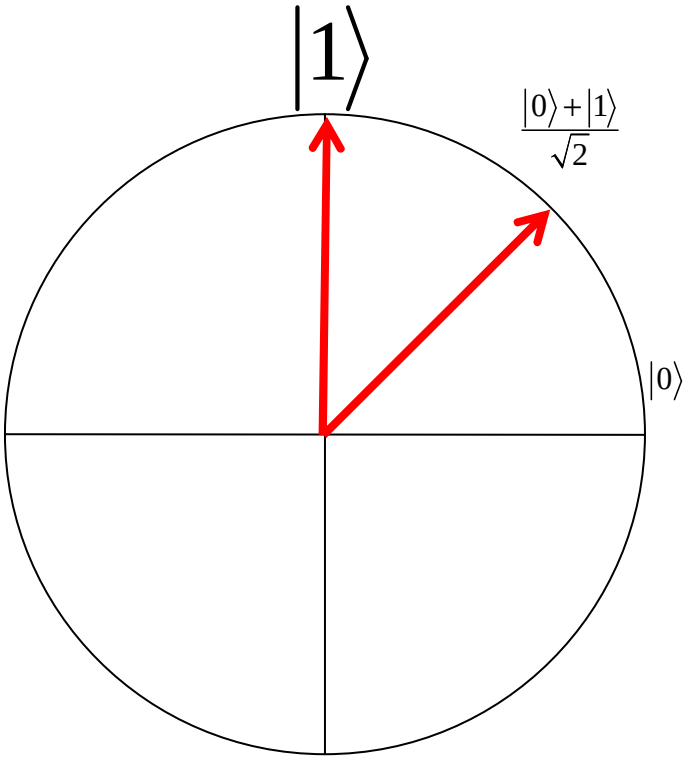
To understand why requires knowing something about what quantum mechanics actually says. But don't let that alarm you. For I'll let you in on a secret: despite claims to the contrary, quantum mechanics is easy! The only complicated part is *applying* the theory to electrons, photons, and other real particles. But to understand quantum computing, you don't need to worry about any of that, any more than you'd have to worry about transistors and voltage potentials when writing a Java program. All you need to know about are certain numbers called "amplitudes," which are a lot like probabilities, except with minus signs.

In ordinary life, you might say there's a 50% or 20% chance of rain tomorrow, but you'd never say there's a *minus* 20% chance. In quantum mechanics, however, any event is assigned a number called an "amplitude," which could be positive or negative. (Actually, amplitudes could also be complex numbers, but that's a detail we'll ignore.) For example, a cat might have a $1/\sqrt{2}$ amplitude of being alive, and a $-1/\sqrt{2}$ amplitude of being dead. Physicists would write this as follows, using the "Dirac ket notation" (which one gets used to with time):

$$|\text{Cat}\rangle = 1/\sqrt{2}\ |\text{Alive}\rangle - 1/\sqrt{2}\ |\text{Dead}\rangle.$$

To find the probability of a given measurement outcome, you take its amplitude and square it. In this case, the cat would be found "alive" with probability $(1/\sqrt{2})^2$ or 1/2, and "dead" with probability $(-1\sqrt{2})^2$, or again 1/2. Also, the cat "snaps" to whichever outcome we see. For example, if we measure the cat to be alive and then measure it again, nothing having happened in the interim, we'll still find it to be alive. Of course, the probability of finding the cat alive and of finding it dead should sum to 100%. (Quantum mechanics is crazy, but not *that* crazy.) This means that the sum of the squares of the amplitudes should be 1, which leads to a set of possible states that forms a circle [see Figure].

But what's the point of talking in such a roundabout way? Why not just say that the cat is alive with probability 50% and dead with probability 50%, but we don't know which until we measure it? This question takes us to the heart of quantum mechanics: the way the amplitudes change if we do something to the cat. In theory, we could perform any operation that rotates or reflects the circle of possibilities in the figure below.



For example, we could perform a 45° counterclockwise rotation, which maps the state |Alive⟩ to 1/√2 |Alive⟩ + 1/√2 |Dead⟩, and |Dead⟩ to -1/√2 |Alive⟩ + 1/√2 |Dead⟩. (In practice, this sort of thing is a lot easier with electrons and photons than with cats.)

If we apply a 45° rotation to the |Alive⟩ state, we get a cat in an equal superposition of |Alive⟩ and |Dead⟩. If we then measure, we find the cat alive with 50% probability and dead with 50% probability. But what happens if we rotate by 45° a second time, without measuring the cat in between rotations? We get an |Alive⟩ cat rotated by 90° toward the |Dead⟩ axis: in other words, a dead cat! So applying a "randomizing" operation to a "random" state produces a "deterministic" outcome. One way to understand this is that, while there are two paths (or if you like, "parallel universes") that lead to the cat being alive (namely, |Alive⟩→|Alive⟩→|Alive⟩ and |Alive⟩→|Dead⟩→|Alive⟩), one of these paths has amplitude ½ and the other has amplitude -½. As a result, the paths interfere destructively and cancel each other out, so all that's left are the paths that lead to the cat being dead [see Figure]. Whenever physicists or science writers refer to the "bizarre laws of the quantum world," this sort of interference between positive and negative amplitudes is ultimately what they're talking about.

The above applied to a single quantum bit or "qubit," with two measurable states: |Alive⟩ and |Dead⟩, |1⟩ and |0⟩. But eventually we hope to build quantum computers involving many thousands of these qubits: not cats, but maybe atomic nuclei that can spin either up or down, or electrons that can be in either a ground state or an excited state. To describe such a computer, we'd have to specify an amplitude for every possible result of measuring the qubits in order. For example, if there were 10,000 qubits, then we'd need

$2^{10,000}$ amplitudes—which, if you think about it, is a pretty staggering amount of information for Nature to keep track of at once. The goal of quantum computing is to exploit this strange sort of parallelism that is inherent in the laws of physics as we currently understand them.

The trouble is that, the instant we look at our computer, we "collapse" it like a soufflé—meaning that we see only one of the $2^{10,000}$ possible measurement outcomes. To switch metaphors, it's as if our computer conjures into existence an exponentially-powerful monster, who then scurries into a closet as soon as we switch on the lights. But in that case, why does it even matter that the monster was ever there? Because in principle, we might be able to choreograph the interference between positive and negative paths, so that the unwanted answers all end up with amplitudes close to 0. If we can pull that off, then the right answer will be observed with overwhelming probability.

A priori, it's not obvious that we can actually use this sort of interference to solve any interesting problem. But, building on work of Daniel Simon and other researchers, in 1994 Peter Shor gave an astonishing quantum-mechanical algorithm to factor integers in polynomial time. The importance of the factoring problem, and therefore of Shor's breakthrough, can be understood by anyone who's ever ordered something over the Internet with a credit card. To prevent your credit card number from being stolen by hackers, your web browser automatically encrypts the number before sending it. But how is it possible for your computer to encrypt a message to (let's say) Amazon.com, if you've never met an Amazon employee in an abandoned garage at 2AM to agree on the key? After all, sending the key over the Internet would defeat the purpose! The solution comes from a revolutionary system called *public-key cryptography*, which was first described to the world in a 1978 *Scientific American* column by Martin Gardner. In this system, first Amazon's server generates both a "public" key and a "private" key, and sends your computer the public key. Then, using the public key, your computer encrypts your credit card number in such a way that in can only be *decrypted* by someone who knows the private key. The most popular such system is called RSA, after Ronald Rivest, Adi Shamir, and Leonard Adleman, who invented it in 1976. (Though as we now know, cryptographers at Britain's GCHQ invented RSA several years earlier, but kept their discovery secret.)

So what's the catch? Well, the security of RSA depends on the assumption that factoring large numbers is harder than multiplying them. For example, it might take you weeks to figure out the prime factors of 17,197,227,206,573 by hand—but if a genie whispered in your ear that they're 4,333,829 and 3,968,137, then it would be easy to multiply these numbers together and see if they check out. (They do.) In RSA, you can think of 17,197,227,206,573 as being the public key, and 4,333,829 and 3,968,137 as being the private key. Then the point is that, by factoring the public key, a hacker could recover the private key and thereby decrypt your credit card number.

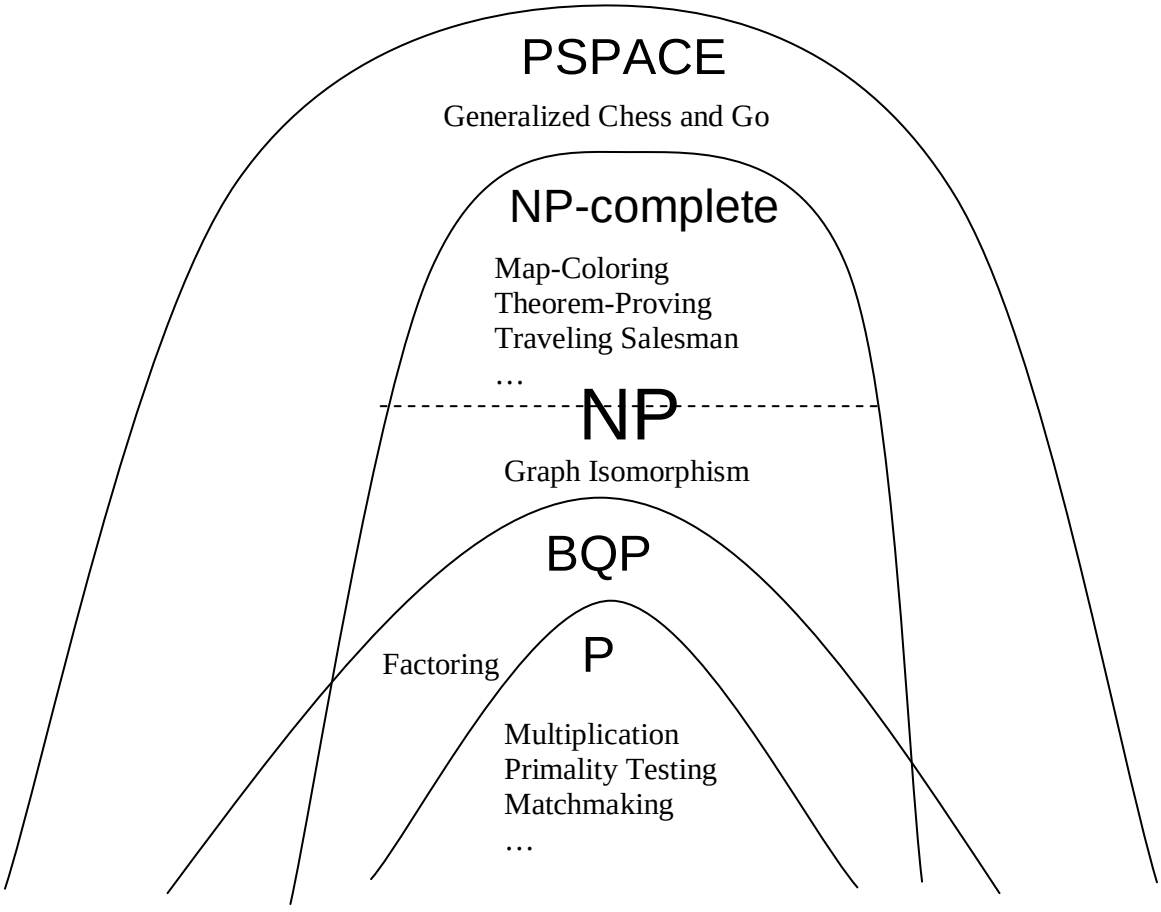
On a conventional computer, the fastest known factoring method uses an amount of time that increases about exponentially with $\sqrt[3]{n}$, where n is the number of digits in the number to be factored. What Shor showed is that a quantum computer could factor an n-digit

number using only about n^2 steps—an exponential improvement. His algorithm works by first reducing the problem to a seemingly unrelated one: that of finding the *period* of an exponentially long sequence of numbers. As an example, the following sequence has a period of 7:

… 9 2 5 6 1 3 8 9 2 5 6 1 3 8 9 2 5 6 1 3 8 9 2 5 6 1 3 8 …

To find the period of such a sequence, the algorithm first generates a vast superposition with one path for every element of the sequence. Then, by using a powerful tool called a quantum Fourier transform, it causes these paths to interfere destructively and cancel each other out, *unless* they lead to information about the period. By repeating the algorithm several times and combining the measurement outcomes, one can then reconstruct the period (and from that the factors) with an arbitrarily small chance of error—say, 0.0001%.

Some people mistakenly think that Shor's algorithm would let us do even more than break RSA—indeed, that it would let us solve NP-complete problems in polynomial time. But alas, while the factoring problem is in NP, it's not known to be NP-complete. Indeed, most computer scientists believe that factoring is one of the rare problems that are "intermediate" between P and NP-complete [see diagram below].



They believe this because factoring has special properties that don't seem to be shared by NP-complete problems. For example, while a map might be colorable in zero, six, or thirty billion ways, any positive integer can be factored into primes in exactly one way. To create his algorithm, Shor relied heavily on such special properties of the factoring problem.

So then, could quantum computers solve NP-complete problems efficiently or not? The answer should come as no surprise by now: we don't know! After all, we don't even know if *classical* computers can solve them—and anything a classical computer can do, a quantum computer could do also (just as you could play Pac-Man on a Sony PlayStation if you wanted to). What we do know is that, if there *were* a fast quantum algorithm to solve NP-complete problems, it would have to rely on more than "brute force." To see what this means, imagine that you're searching for a gold coin hidden in one of N boxes, and that you have no clues about its location: all you can do is pick a box, open it, and see if the coin is inside. Then clearly you'll have to open about half the boxes on average before you find the coin. But what if you could open *all N boxes* in superposition, then cause the paths in the superposition to interfere, then open all N boxes in superposition again, and so on? Even in that fanciful scenario, you'd *still* need at least $\sqrt{N}$ steps to find the coin, according to a 1994 theorem of Charles Bennett, Ethan Bernstein, Gilles Brassard, and Umesh Vazirani. Shortly after this finding, Lov Grover showed that the $\sqrt{N}$ bound is achievable: in other words, there *exists* a quantum-mechanical algorithm to find the coin after only about $\sqrt{N}$ steps. Amusingly, Grover's algorithm was proven to be optimal before it was discovered to exist!

What Bennett et al.'s theorem means is that, in searching for a needle in a haystack, quantum computers would offer a small "generic" advantage, but would not slay the ogre of intractability. Where a classical computer would need 2^{1000} steps to search 2^{1000} possible solutions, a quantum computer would need "merely" the square root of that, or 2^{500}. Ultimately, the reason is that Schrödinger's equation, which governs how quantum-mechanical amplitudes change in time, is a *linear* equation. If we think of the paths in a superposition as "parallel universes," then this linearity is what prevents the universe that happened to find the answer from simply "telling all the other universes about it."

By the time I came to quantum computing, in 1998, all of this was already old hat. The challenge was to create a general *theory* of when quantum computers outperform classical ones and when they don't. For example, how big an advantage would a quantum computer have in games of strategy such as chess? After my junior year at Cornell, I spent a summer attacking that question unsuccessfully. My goal was to show that, as in the case of NP-complete problems, any quantum algorithm that examined all sequences of moves by "brute force" would need at least the square root of the number of steps needed by a classical algorithm.

Soon after I'd given up, a Berkeley graduate student named Andris Ambainis succeeded with a brilliant new approach, one based on the quantum phenomenon of "entanglement." Instead of considering each game separately, Ambainis imagined feeding the quantum algorithm a superposition over all possible games. He then showed that, if the algorithm

succeeded too quickly, then the entanglement between that superposition and the algorithm's memory would increase at an impossibly fast rate. By now, Ambainis's approach, which he dubbed the "quantum adversary method," has been applied to dozens of other problems.

But ironically, the approach failed on the one problem for which it was originally created. In modern cryptanalysis, one of the most basic tasks is called *collision-finding*: given a function that digitally "signs" email messages, find two messages that get assigned the same signature. If a fast and general quantum algorithm existed to find these "collisions," then there'd *really* be no hope for secure Internet communications in a world with quantum computers.

If there are N possible signatures, then even a classical computer only needs to examine about $\sqrt{N}$ messages before it finds two messages with the same signature. The reason is related to the well-known "birthday paradox": if you put 23 people in a room, then there are better than even odds that two of the people share a birthday, since what matters is the number of *pairs* of people. In 1997, Gilles Brassard, Peter Høyer, and Alain Tapp showed that a quantum computer could beat this "birthday bound," finding a collision after only $\sqrt[3]{N}$ steps instead of $\sqrt{N}$. But is that the best possible? Astonishingly, after five years of effort, researchers were still unable to rule out a quantum algorithm that would use a *constant* number of steps, independent of N!

What made the problem so hard to analyze was its "global" nature. In the case of NP-complete problems, Bennett et al. were able to argue that, if we start out with N empty boxes, and then put a gold coin in one them, a quantum algorithm would "notice" this change with only a tiny amplitude. But a quantum algorithm for collision-finding wouldn't *need* to notice such local changes, since there are numerous collisions, and the algorithm only has to find one of them. Yet in late 2001, after four months of failed attempts, I could finally show that any quantum algorithm for collision-finding would need at least $\sqrt[5]{N}$ steps. My proof argued, first, that a faster algorithm would lead to a certain kind of low-degree polynomial to "distinguish" between many and few collisions; and second, that such a polynomial would violate a theorem proven in 1890 by the Russian mathematician Andrei A. Markov. In a double irony, this polynomial approach was the same one that had failed on the game-playing problem, before Ambainis's entanglement-based approach succeeded! Thanks to subsequent improvements by Yaoyun Shi and others, we now know that any quantum algorithm to find collisions needs at least $\sqrt[3]{N}$ steps, and therefore that Brassard, Høyer, and Tapp's original algorithm was indeed optimal.

Though many other questions about the power of quantum computers remain, by now a tentative picture has emerged. By exploiting the bizarre phenomenon of interference, a quantum computer really *could* achieve dramatic speedups for a few problems, like factoring integers. But for the problems that really excite the imagination—like searching for mathematical proofs or programs that predict the stock market—even a quantum computer would quickly run up against the wall of intractability. If that wall

can ever be breached, it seems like we'll need an entirely new type of computer—one that would make quantum computers look pedestrian by comparison.

Beyond Quantum Computers

If quantum mechanics led to such a striking new model of computation, then what about that other great theory of 20^{th}-century physics, relativity? The idea of "relativity computing" is simple: first start a computer working on a difficult problem; then leave the computer on Earth, board a spaceship, and accelerate to nearly the speed of light. When you return, all of your friends will be long dead, but the answer to your problem will await you.

What's the flaw in this proposal? Well, consider the *energy* needed to accelerate to relativistic speed. In order to achieve an exponential computational speedup, it turns out that you'd have to get exponentially close to the speed of light. But to do that, you'd need an exponential amount of energy—so your fuel tank, or whatever else is powering your spaceship, would have to be exponentially large. But this again means that you'd need an exponential amount of time, just for the fuel from the far parts of the tank to affect you!

Similar problems have long plagued proposals for "hypercomputing," or accelerating computers by an unlimited amount. For example, if you didn't know anything about physics, it'd be easy to imagine performing the first step of a computation in one second, the second step in half a second, the third step in a quarter-second, and so on. As in Zeno's paradoxes, after two seconds you would have performed an infinite number of steps. Why doesn't that work? Because most physicists think that there's a minimum unit of time: the *Planck time*, or about 10^{-43} seconds. If you tried to build a clock that ticked faster than that, you'd use so much energy that the clock would collapse to a black hole. Thought experiments involving black holes also led physicists to the so-called *holographic principle*, which states that the maximum number of bits that can be stored in any region of space is at most proportional to the surface area of the region, at a rate of one bit per "Planck area," or 1.4×10^{69} bits per square meter. Whether these results mean that space and time are literally *discrete* at that scale is hotly debated. But it's already clear that, unless the results are dramatically wrong, we can never achieve an exponential speedup by means of analog or "hyper" computers.

This provides one example of how thinking about the hardness of computational problems leads us straight to the frontier of physics. Ideally, of course, we'd like a "grand unified theory" that would tell us once and for all which computations can be efficiently performed in the physical world. So then what about the current contenders for a quantum theory of gravity, such as string theory and loop quantum gravity? What do they say? Unfortunately, these theories don't yet seem to be mathematically rigorous enough to be turned into models of computation. Indeed, what would seem to an outsider like extremely basic questions—like "what are the possible states?," and "how do they evolve in time?"—are still far from settled. The one clear result is due to Michael Freedman, Alexei Kitaev, and Zhenghan Wang, who showed that topological quantum

field theories—particularly simple "toy" models of quantum gravity, involving only two space dimensions and one time dimension—are equivalent in power to ordinary quantum computers.

But if our goal is to understand what it would take to solve NP-complete problems in polynomial time, then, why let reality get in the way of a good yarn? Why not *assume* (say) the possibility of time travel into the past, and see what happens? The idea was well explained by the movie *Star Trek IV: The Voyage Home*. The Enterprise crew has traveled back in time to the present (meaning to 1986) in order to find humpback whales and transport them to the twenty-third century. The trouble is that building a tank for the whales requires a type of plexiglass that hasn't yet been invented. So the crew seeks out the company that *will* invent the plexiglass, and reveals its molecular formula to that company. The question is, who (or what) invented the formula in the first place?

In a 1991 paper, David Deutsch pointed out that we could use a similar idea to solve NP-complete problems in polynomial time. We'd simply guess a possible solution; then, if the solution didn't work, we'd go back in time and guess the next solution, and so on, until we found a solution that did work—at which point we'd go back in time and guess that *same* solution. Provided a solution exists at all, the only self-consistent outcome (that is, the only outcome that avoids the classic paradoxes, such as going back in time and killing your own grandfather) is the one in which you "happen" to guess a solution that works. Subsequently Dave Bacon studied these "time-travel computers" in greater detail, and showed that, just like ordinary quantum computers, they could in principle tolerate a small amount of noise. Also, I extended Deutsch's time-travel "algorithm" beyond even NP-complete problems, to the so-called PSPACE-complete problems (such as computing an optimal move in chess).

All of this requires the somewhat problematic assumption that time travel is possible at all! It's tempting to conjecture that this is not an accident: indeed, that *any* method for solving NP-complete problems in polynomial time would be found to contain some "unphysical" element. But I'd go even further, and suggest that the intractability of NP-complete problems might eventually be seen as a basic principle of physics. In other words, not only are "closed timelike curves" impossible, but one *reason* why they're impossible is that if they weren't, then we could solve NP-complete problems in polynomial time! I know this seems circular, but it's really no different from two other limitations on technology that emerged from physics: the Second Law of Thermodynamics, and the impossibility of faster-than-light communication. As with the hardness of NP-complete problems, we believe that these principles are true because we've never seen a counterexample, and we're skeptical of proposed counterexamples because we believe the principles are true.

Of course, there is *one* fast and reliable method to solve NP-complete problems in polynomial time: first generate a random solution, then kill yourself if the solution is incorrect. But short of tying your own existence to a computer's output, I believe that the world we inhabit contains no royal road to creativity, no magic sieve to cull the archangels' autobiographies from the Library of Babel.