from *SIAM News*, Volume 36, Number 1, January/February 2003

Emerging Insights on Limitations of Quantum Computing Shape Quest for Fast Algorithms

By Sara Robinson

Read a typical article about quantum computing (including this one) and almost surely, somewhere in the first three paragraphs, you'll find a reference to Peter Shor's polynomial-time quantum algorithm for factoring.

Shor's 1994 result propelled quantum computing into the limelight, attracting hordes of eager young researchers and abundant government funding. If a quantum computer could factor in polynomial time, computer scientists theorized, perhaps it could solve many other important, seemingly intractable problems.

Eight years later, these hopes have not been realized. The smattering of quantum algorithms that followed Shor's are for lesser problems than factoring and don't have significant applications. To be sure, work on equally important but less flashy aspects of quantum computing, such as error correction and fault tolerance, has moved forward at a rapid clip. Still, quantum algorithms for important problems have so far eluded the best efforts of the hordes.

Asked why progress has been so slow, quantum computing researchers point to the successes in another area of quantum computing research: quantum lower bounds. A series of recent results indicate that while quantum computers are more powerful than their classical counterparts, that power has significant limitations. By proving lower bounds to the run-time for certain types of quantum algorithms, researchers have ruled out whole classes of important problems as likely candidates for fast quantum algorithms. Only a few prominent problems remain.

"We've realized that the target problems for quantum algorithms are few and far between," said Alex Russell, a professor of computer science at the University of Connecticut, during a fall visit to the Mathematical Sciences Research Institute for a special semester on quantum computing.

First to be scratched off as plausible targets were the NP-complete problems, the class of problems that have been shown to be the hardest in NP. In a 1994 result announced just before Shor's, Umesh Vazirani, a professor of computer science at the University of Calfornia, Berkeley, and colleagues demonstrated that a quantum computer is unlikely to overcome the barriers that have stymied attempts to build a classical algorithm for these problems.

More recently, researchers have shown that the same holds true for some problems used in cryptography. Such has been the fate of some of the best candidates for showcasing dramatic quantum speedups. Still, an understanding of the limitations of quantum computing has been helpful to the quest for new quantum algorithms.

"What we learned about NP-complete problems was a big setback, but now we have a better idea of which problems to focus on and what techniques couldn't possibly work," says Vazirani, the organizer of MSRI's fall 2002 program on quantum computing.

A Bit of Background

A "classical" bit is typically represented by an electrical current or voltage that is either high or low. Thus, classically stored information can easily be read, corrected, or copied. A quantum computer, however, makes use of one of nature's more sophisticated storage systems: the states of elementary particles.

Thinking classically, the nucleus of a hydrogen atom, say, is either aligned with a magnetic field or it is not. But in the weird world of quantum physics, the state of the electron is not just high or low but a weighted combination of the two: what physicists call a "superposition" of classical states. Mathematically speaking, a quantum bit is a unit vector in C (written $\alpha|0\rangle + \beta|1\rangle$, where α, β in C are the *amplitudes* of $|0\rangle$ and $|1\rangle$, respectively). A two-bit state, a superposition of four possible classical configurations, is thus a unit vector in C^4, written $\alpha|00\rangle + \beta|01\rangle + \gamma|10\rangle + \delta|11\rangle$). In general, as the number of quantum bits grows, the amount of information required to describe the system grows exponentially. In some sense, then, a quantum system contains more information than its classical counterpart.

At first glance, it might seem that a quantum computer could quickly solve any of the computationally challenging problems that make up the class known as NP. But nature's vast storehouse of information proves difficult to tap. According to the theory of quantum mechanics, the act of observing a quantum state jars it out of its delicate superposition, collapsing it into a classical state with probability that's the square of the amplitude of the classical state.

A computation with a quantum computer must follow the rule of reversibility for quantum physics. Thus, a quantum algorithm consists of a sequence of unitary transformations applied to an initial quantum state. The last step is a measurement. The trick to designing a quantum algorithm is to set up a final state where, if you perform a measurement, you get the right answer with high probability.

Quantum algorithms make use of the elusive phenomenon known as *quantum entanglement*, mysterious correlations between particles that cannot be explained by classical physics. The two-bit quantum state $1/\sqrt{2}|00\rangle + 1/\sqrt{2}|11\rangle$, for instance, has the property that if the first bit is measured, then the second bit must have the same value.

Complexity Theory

The goal of devising quantum algorithms is to say something about complexity theory, the framework used by computer scientists to categorize computational problems.

Theorems of complexity theory—even classical ones—look funny to the uninitiated. In their attempts to classify algorithms by their computational "hardness," theoretical computer scientists kept bumping up against very hard yet fundamental problems that they weren't able to solve. The only way to make progress around these roadblocks was to resort to proving lots of conditional statements. Computer scientists can say, for instance, that "if this problem is hard, so is this one," but they typically cannot make definitive statements about the computational hardness of a problem unless it's been shown to be easy. "Easy" means that there's an algorithm that gives the right answer in time polynomial in the length of the input, putting the problem in the complexity class P.

Also "easy" are problems for which an algorithm involving some kind of coin flip gives the correct answer with high probability. Such problems are said to be in BPP, the class of problems for which there is a (probabilistic) polynomial-time algorithm that gives the right answer at least two thirds of the time.

A precise definition of NP is tricky; basically, the NP class includes a collection of decision (yes or no) problems—e.g., Is this number composite?—for which a "yes" answer can be verified in polynomial time. Within NP are the NP-complete problems, a polynomial-time algorithm for any one of which would yield a polynomial-time algorithm for all the problems in NP—and would prove that P = NP.

Complexity theorists know that P is contained both in NP and in BPP, but it's not known whether P = NP or whether P = BPP. It's also not known whether BPP contains NP or vice-versa. All that is known is that both classes fall within PSPACE, the set of problems computable with polynomial space on a Turing machine tape.

So where do quantum algorithms fit into this morass? In 1993, Vazirani and his student Ethan Bernstein defined the complexity class BQP (for bounded-error, quantum polynomial time) as problems for which there is a quantum algorithm that gives the right answer at least two thirds of the time; they showed that BQP sits between BPP and PSPACE (see Figure 1).

What computer scientists really wanted to know, however, was whether the enormous information resources a quantum computer draws upon are enough to solve an NP-complete problem. Based on a 1997 paper by Bernstein and Vazirani, together with Charles Bennett and Gilles Brassard, CS theorists now think not.

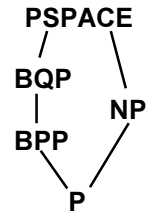

Figure 1. *Defined in 1993, the complexity class BQP (bounded error, quantum polynomial time) lies between BPP and PSPACE.*

An Almost Lower Bound for NP-hard Problems

The canonical NP-complete problem, known as SAT, is to figure out whether there's a satisfying assignment for a formula $f(x_1, \ldots x_n)$, in n Boolean variables. As with the other hard problems in NP, the challenge is the exponentially large space of possible candidates to check (each n-bit string being a potential satisfying assignment), given that the only technique for finding the right one is to try all the possibilities, one after the other.

The initial promise of quantum computing was that all the possibilities could be placed in superposition and tried simultaneously; somehow, the final information would then be manipulated so that a measurement would yield the correct answer with high probability. However, Bernstein, Vazirani, and their colleagues showed that this "brute force" approach wouldn't work.

To get their result, the researchers assume that $f(x_1, \ldots x_n)$ is given as a "black box": It can be evaluated on any input $x_1, \ldots x_n$ in a single step, but the formula itself is not available for inspection. Of course, a quantum algorithm is allowed to evaluate f on any superposition of all (or any subset of) the exponential-ly many choices for $x_1, \ldots x_n$ in a single step. This is called a (quantum) query.

A black box argument doesn't actually prove a lower bound—it may be that a quantum algorithm could exploit the structure of SAT in a sophisticated way that goes well beyond this black box model. If so, however, it's hard to think of a way of doing this that a classical algorithm could not exploit as well. In any case, black box bounds are the only feasible ones, pending resolution of long-standing open problems of complexity theory. (Proving that there are problems in NP that are not in BQP, for instance, would show that P is different from NP.)

Any algorithm that solves "black box SAT" has to be able to tell the difference, in particular, between an unsatis-fiable Boolean formula and a uniquely satisfiable one. What the researchers have shown is that any algorithm that does this must fail to give the correct answer for some formula, unless it makes exponentially many queries.

The proof consists of first performing a thought experiment to figure out the weak point of the algorithm: The researchers imagine a test run on an unsatisfiable formula f to determine the assignment x for the variables that is queried the least often in the algorithm (i.e., the sum of the squares of the amplitudes with which it is queried is no larger than for any other assignment in the algorithm). Certainly, this is no more than the average, which is $T/2^n$, where T is the number of queries.

They then consider what the algorithm does on a formula g_x with unique satisfying assignment x. The idea is that since x is the least frequently queried assignment, the final state of the algorithm should be practically identical, regardless of whether the input is u or g_x. To prove this formally, the researchers invoke a hybrid argument—they morph the algorithm applied to u into the algorithm applied to g_x, one query at a time, and show that the total chance in the final state of the algorithm is bounded by the sum of the absolute values of the amplitudes with which x is queried.

Thus, to make the answer measurable with a probability bounded away from zero, there have to be quite a few steps: at least $\sqrt{2^n}$ of them in the worst case, since each amplitude with which x is queried is $1/\sqrt{2^n}$. Amazingly, this bound is tight; Lov Grover's search algorithm realizes it. (See sidebar on page 3.)

This bound is not a proof that all of NP is out of reach, but it is a demonstration that the obvious advantage of a quantum computer isn't enough to span the exponential gap.

Lower Bounds for One-way Functions

Another fundamental computational problem that is especially dear to cryptographers is to invert a randomly chosen permutation π of the numbers 1 through 2^n. That is, given y, find x so that $\pi(x) = y$. This problem is the canonical model of a one-way function—easy to compute but hard to undo. Although such functions have not been rigorously proven to exist, the assumption that they do is the basis of many cryptosystems.

In the RSA cryptosystem, for example, cubing modulo N plays the role of a one-way function. Shor's algorithm for factoring, if ever realized on a large enough scale, would make finding cube roots easy and would thus break RSA.

Thus, Andris Ambainis's 1999 lower bound on inversion of permutations was heartening to cryptographers because it leaves open the possibility of one-way functions for quantum computers. (Ambainis was to have spent the fall semester as an MSRI postdoc but remained in his native Latvia, having been denied a visa to visit the U.S.) The hybrid argument, applied to the permutation inversion problem, yields a lower bound of only $\sqrt[3]{2^n}$. Again, making a black box assumption about the algorithm, Ambainis was able to obtain a bound of $\sqrt{2^n}$ and to show that his bound is tight.

To prove his bound, Ambainis developed what is now called the "adversary method," which is based on physical intuition: the notion of quantum entanglement. He imagined running a quantum algorithm for the inversion problem on a superposition of inputs and noted that the state of the input together with the algorithm workspace would start out completely unentangled yet end up highly entangled. He then showed that, in each query, the entanglement increases by no more than $1/\sqrt{n}$. Thus, the number of queries in the algorithm is at least $\sqrt{2^n}$.

A Sample Algorithm: Quantum Search

Suppose you want to find a single entry in an N-element database. Algorithmically speaking, that means you have a function $f(k)$ that is guaranteed to be nonzero at precisely one value of k, between 1 and N. Classical computation has no recourse but to compute $f(k)$ for one value of k after another, so that the average search requires $N/2$ calculations. It would seem that you can't do better than $N/2$, but that's classical thinking. In 1996, Lov Grover showed that you could do it with only $O\sqrt{N}$ calculations if you happened to have access to a quantum computer.

Grover's algorithm takes advantage of the linear-algebraic structure of quantum mechanics to achieve "amplitude amplification." Starting with a uniform superposition of all strings of length 2^N, Grover does the following sequence of operations $\sqrt{N}$ times:

First, he applies a unitary operator U_m, where U_m is defined as

$$U_m|n\rangle = -|n\rangle \text{ for } n = m,$$
$$U_m|n\rangle = |n\rangle \text{ otherwise.}$$

Then he applies another unitary operator, which has the effect of reflecting all the amplitudes about their mean. Because the sign of the desired amplitude has been reversed, the operation amplifies this amplitude at the expense of the others. After $\sqrt{N}$ repetitions, this amplitude is bounded away from zero, which is enough to guarantee that you can measure it.

Hash Functions

Authentication schemes in cryptography often depend on many-to-one functions known as hash functions. A good cryptographic hash, h, has the property that given x, $h(x)$, it's hard to find a y such that $h(x) = h(y)$.

The essential features of this problem are abstracted in the *collision problem*: Given a function f on $1 \ldots 2^n$ that's guaranteed to be one-to-one or two-to-one, decide which of the two it is. A lower bound of $\sqrt[5]{2^n}$ queries was found earlier this year by Scott Aaronson, a student of Vazirani's.

Aaronson's proof technique is an extension of a clever method using polynomials first developed by Beals, Buhrman, Cleve, Mosca, and de Wolf in 1998. This group showed that if a problem given by a black box function $f(x_1, \ldots x_n)$ can be resolved by a quantum algorithm making k queries, then it can be uniformly approximated by a polynomial of degree no higher than $2k$. A quantum lower bound of L follows from the proof that any polynomial of degree less than or equal to L does not approximate the target function. In general, unfortunately, it is very hard to prove such lower bounds on the degree of an approximating polynomial. If the target function is very symmetrical, however, the essential features of the problem can be projected down to a single dimension. Since, by Markov's inequality, the degree of a univariate polynomial is proportional to its maximum derivative, you can obtain the desired lower bound.

Although the collision problem could not be reduced to one dimension, Aaron-son realized that the probability that the algorithm outputs one on a typical d-to-one function (i.e., chosen uniformly at random among all d-to-one functions) is itself a polynomial in d (and another variable); moreover, the degree of this polynomial is at most $2k$ if the algorithm makes k queries. Because the algorithm must distinguish one-to-one from two-to-one functions, the polynomial must have a large derivative somewhere between $d = 1$ and $d = 2$; but because the polynomial expresses the probability of a certain event, it is also bounded between 0 and 1 for all values of d. A suitable modification of Markov's inequality can now be applied to prove the lower bound.

What's Next?

Now that computer scientists have shown NP-complete problems, one-way functions, hash functions, and others to be resistant to brute-force quantum attacks, what's left? The leading candidate seems to be graph isomorphism: a long-standing open problem that, like factoring, is not known either to be in P or to be NP-complete.

A quantum polynomial-time algorithm for this problem would make quite a splash—let's hope that a lower bound doesn't come first.

Sara Robinson is a freelance writer based in Pasadena, California.