6.080/6.089 GITCS May 13, 2008

Lecture 24

Lecturer: Scott Aaronson *Scribe: Chris Granade*

1 Quantum Algorithms

Of course the real question is: can quantum computers actually do something more efficiently than classical computers? In this lecture, we'll see why the modern consensus is that they can.

1.1 Computing the XOR of Two Bits

We'll first see an algorithm due to Deutsch and Jozsa. Even though this algorithm is trivial by modern standards, it gave the first example where a quantum algorithm could provably solve a problem using fewer resources than a classical algorithm.

Suppose we're given access to a Boolean function $f : \{0,1\} \rightarrow \{0,1\}$. And suupose we want to compute $f(0) \oplus f(1)$, the XOR of $f(0)$ and $f(1)$. Classically, how many times would we need to evaluate f? It's clear that the answer is twice: knowing only $f(0)$ or $f(1)$ tells us exactly nothing about their XOR.

So what about in the quantum case? Well, first we need to say what it even *means* to evaluate f. Since this is a quantum algorithm we're talking about, we should be able to evaluate both inputs, $f(0)$ and $f(1)$ in quantum superposition. But we have to do so in a reversible way. For example, we can't map the state $|x, b\rangle$ to $|x, f(x)\rangle$ (overwriting b), since that wouldn't be unitary.

The standard solution is that querying f means applying a unitary transformation that maps $|x, y\rangle \rightarrow |x, y \oplus f(x)\rangle$. Is it reversible? Yeah. Applying it twice gets you back to where you started. I claim we can compute $f(0) \oplus f(1)$ using just a single one of these operations. How?

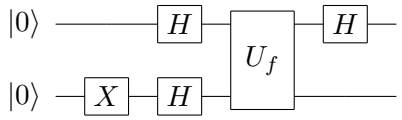

Figure 1: Finding $f(0) \oplus f(1)$ in one query.

In the circuit above, the effect of the gates before U_f is to prepare an initial state $|\psi_0\rangle$:

$$|\psi_0\rangle = |+\rangle\,|-\rangle = \frac{1}{2}\left[|0\rangle + |1\rangle\right]\left[|0\rangle - |1\rangle\right]$$

If you think of the effect of U_f on the first qubit in this state, it's just to negate the amplitude if $f(0) \neq f(1)$! Thus, U_f produces $|+\rangle\,|-\rangle$ if $f(0) = f(1)$ and $|-\rangle\,|-\rangle$ otherwise. The final Hadamard gate transforms the first qubit back into the computational basis, so that we measure 1 if and only if $f(0) \neq f(1)$.

In particular, this means that if you want to compute the XOR of N bits with a quantum computer, you can do so using $N/2$ queries, as follows: first divide the bits into $N/2$ pairs of bits,

then run the above algorithm on each pair, and finally output the XOR of the results. Of course, this is only a constant-factor speedup, but it's a harbinger of much more impressive speedups to come.

1.2 Simon's Algorithm

Say you're given a Boolean function $f : \{0,1\}^n \rightarrow \{0,1\}^n$. You're promised there exists a "secret string" s such that $f(x) = f(y)$ if and only if $y = x \oplus s$, where $\oplus$ denotes a sum mod 2. The problem is to find s by querying f as few times as possible.

How many queries would a classical randomized algorithm need to solve this problem? Something very similar was on your problem set! Right, $2^{n/2}$. This is basically just the birthday paradox. Until it happens to find an x, y pair such that $f(x) = f(y)$, your algorithm is basically just "shooting in the dark"; it has essentially no information about s. And after T queries, the probability of having found an x, y pair such that $f(x) = f(y)$ is at most $T^2/(2^n - 1)$ (why?).

On the other hand, in 1993 Daniel Simon gave a quantum algorithm that solves this problem in polynomial time, in fact using only $O(n)$ queries. This was the first example of a problem that a quantum computer can solve exponentially faster than a classical one. Admittedly, it's a contrived example (and probably for that reason, Simon's paper was originally rejected!). But it's good to see for two reasons: first, it led directly to Shor's factoring algorithm. And second, the easiest way to *understand* Shor's algorithm is to understand Simon's algorithm, and then see Shor's algorithm as the same thing with a different underlying group!

Before proceeding further, though, there's one thing I want to clear up. I said that Simon's problem was the first known example where quantum computers provably give an exponential speedup over classical computers. How is that consistent with what I said before, that we can't prove $\mathsf{P} \neq \mathsf{BQP}$ unconditionally?

Right, Simon's problem involves the function f as a "black-box." In the black-box setting, we *can* prove unconditionally that quantum computers give an exponential speedup over classical ones.

1.3 RSA

Alright, so let's say you want to break the RSA cryptosystem, in order to rob some banks, read your ex's email, whatever. We all know that breaking RSA reduces to finding the prime factors of a large integer N. Unfortunately, we also know that "trying all possible divisors in parallel," and then instantly picking the right one, isn't going to work. Hundreds of popular magazine articles notwithstanding, trying everything in parallel just isn't the sort of thing that a quantum computer can do. Sure, in some sense you can "try all possible divisors" – but if you then measure the outcome, you'll get a random potential divisor, which almost certainly won't be the one you want.

What this means is that, if we want a fast quantum factoring algorithm, we're going to have to exploit some *structure* in the factoring problem: in other words, some mathematical property of factoring that it *doesn't* share with just a generic problem of finding a needle in a haystack.

Fortunately, the factoring problem has oodles of special properties. What are some examples we discussed in class? Right: if I give you a positive integer, you might not know its prime factorization, but you do know that it has exactly *one* factorization! By contrast, if I gave you (say) a Sudoku puzzle and asked you to solve it, *a priori* you'd have no way of knowing whether it had exactly one solution, 200 million solutions, or no solutions at all. Of course, knowing that there's exactly one needle in a haystack is still not much help in finding the needle! But this uniqueness is a hint that

the factoring problem might have *other* nice mathematical properties lying around for the picking. As it turns out, it does.

The property we'll exploit is the reducibility of factoring to another problem, called period-finding. OK, time for a brief number theory digression. Let's look at the powers of 2 mod 15:

$$2,\ 4,\ 8,\ 1,\ 2,\ 4,\ 8,\ 1,\ 2,\ 4,\ \ldots$$

As you can see, taking the powers of 2 mod 15 gives us a *periodic sequence*, whose period (i.e., how far you have to go before it starts repeating) is 4. For another example, let's look at the powers of 2 mod 21:

$$2,\ 4,\ 8,\ 16,\ 11,\ 1,\ 2,\ 4,\ 8,\ 16,\ \ldots$$

This time we get a periodic sequence whose period is 6.

What's a general rule that governs what the period will be? We discussed this earlier, when we were talking about the RSA cryptosystem! The beautiful pattern, discovered by Euler in the 1760s, is this. Let N be a product of two prime numbers, p and q, and consider the sequence:

$$x \bmod N,\ x^2 \bmod N,\ x^3 \bmod N,\ x^4 \bmod N,\ \ldots$$

Then, provided that x is not divisible by p or q, the above sequence will repeat with some period that divides $(p-1)(q-1)$. So, for example, if $N = 15$, then the prime factors of N are $p = 3$ and $q = 5$, so $(p-1)(q-1) = 8$. And indeed, the period of the sequence is 4, which divides 8. If $N = 21$, then $p = 3$ and $q = 7$, so $(p-1)(q-1) = 12$. And indeed, the period is 6, which divides 12.

Now, I want you to step back and think about what this means. It means that *if* we can find the period of the sequence of powers of $x \bmod N$, *then* we can learn something about the prime factors of N. In particular, we can learn a divisor of $(p-1)(q-1)$. Now, I'll admit that's not as good as learning p and q themselves, but grant me that it's something. Indeed, it's more than something: it turns out that if we could learn several random divisors of $(p-1)(q-1)$ (for example, by trying different random values of x), then with high probability we could put those divisors together to learn $(p-1)(q-1)$ itself. And once we knew $(p-1)(q-1)$, we could then use some more little tricks to recover p and q, the prime factors we wanted. (This is again in your problem set.)

So what's the fly in the ointment? Well, even though the sequence of powers mod$N$ will *eventually* start repeating itself, the number of steps before it repeats could be almost as large as N itself – and N might have hundreds or thousands of digits! This is why finding the period doesn't seem to lead to a fast *classical* factoring algorithm.

Aha, but we have a quantum computer! (Or at least, we're *imagining* that we do.) So maybe there's still hope. In particular, suppose we could create an enormous quantum superposition over all the numbers in our sequence:

$$\sum_r |r\rangle\,|x^r \bmod N\rangle$$

Then maybe there's some quantum operation we could perform on that superposition that would reveal the period.

The key point is that we're no longer trying to find a needle in an exponentially-large haystack, something we *know* is hard even for a quantum computer. Instead, we're now trying to find the

period of a sequence, which is a *global* property of all the numbers in the sequence taken together. And that makes a big difference.

Look: if you think about quantum computing in terms of "parallel universes" (and whether you do or don't is up to you), there's no feasible way to detect a *single* universe that's different from all the rest. Such a lone voice in the wilderness would be drowned out by the vast number of suburb-dwelling, Dockers-wearing conformist universes. What one can hope to detect, however, is a joint property of *all* the parallel universes together – a property that can only be revealed by a computation to which all the universes contribute [1].

So, the task before us is not hopeless! But if we want to get this period-finding idea to work, we'll have to answer two questions:

1. Using a quantum computer, can we quickly create a superposition over $x \bmod N$, $x^2 \bmod N$, $x^3 \bmod N$, ...?

2. Supposing we did create such a superposition, how would we figure out the period?

Let's tackle the first question first. We can certainly create a superposition over all integers r, from 1 up to N^2 or so. The trouble is, given an r, how do we quickly compute $x^r \bmod N$? We've already seen the answer: repeated squaring!

OK, so we can efficiently create a quantum superposition over all pairs of integers of the form $(r, x^r mod N)$, where r ranges from 1 up to N or so. But then, given a superposition over all the elements of a periodic sequence, how do we extract the period of the sequence?

Well, we've finally come to the heart of the matter – the one part of Shor's quantum algorithm that actually depends on quantum mechanics. To get the period out, Shor uses something called the *quantum Fourier transform*, or QFT. My challenge is, how can I explain the QFT to you without going through the math? Hmmmm...

OK, let me try this. Like many computer scientists, I keep extremely odd hours. You know that famous experiment where they stick people for weeks in a sealed room without clocks or sunlight, and the people gradually shift from a 24-hour day to a 25- or 26- or 28-hour day? Well, that's just ordinary life for me. One day I'll wake up at 9am, the next day at 11am, the day after that at 1pm, etc. Indeed, I'll happily 'loop all the way around' if no classes or appointments intervene.

Now, here's my question: let's say I tell you that I woke up at 5pm this afternoon. From that fact alone, what can you conclude about how long my "day" is: whether I'm on a 25-hour schedule, or a 26.3-hour schedule, or whatever?

The answer, of course, is not much! I mean, it's a pretty safe bet that I'm not on a 24-hour schedule, since otherwise I'd be waking up in the morning, not 5pm. But almost any other schedule – 25 hours, 26 hours, 28 hours, etc. – will necessarily cause me to "loop all around the clock," so that it'd be no surprise to see me get up at 5pm on some particular afternoon.

Now, though, I want you to imagine that my bedroom wall is covered with analog clocks. These are very strange clocks: one of them makes a full revolution every 17 hours, one of them every 26 hours, one of them every 24.7 hours, and so on for just about every number of hours you can imagine. (For simplicity, each clock has only an hour hand, no minute hand.) I also want you to imagine that beneath each clock is a posterboard with a thumbtack in it. When I first moved into my apartment, each thumbtack was in the middle of its respective board. But now, whenever I

[1] For safety reasons, please don't explain the above to popular writers of the "quantum computing = exponential parallelism" school. They might shrivel up like vampires exposed to sunlight.

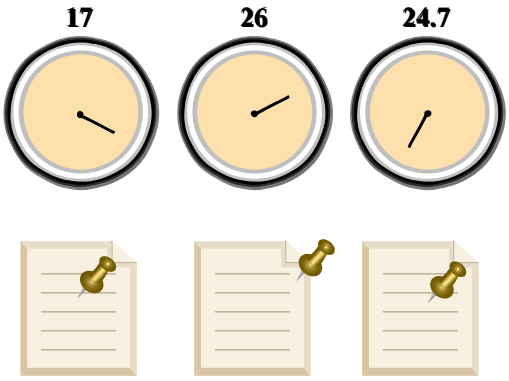

Figure 2: A possible configuration of clocks and pegboards.

wake up in the "morning," the first thing I do is to go around my room, and *move each thumbtack exactly one inch in the direction that the clock hand above it is pointing.*

Now, here's my new question: *by examining the thumbtacks in my room, is it possible to figure out what sort of schedule I'm keeping?*

I claim that it *is* possible. As an example, suppose I was keeping a 26-hour day. Then what would happen to the thumbtack below the 24-hour clock? It's not hard to see that it would undergo periodic motion: sure, it would drift around a bit, but after every 12 days it would return to the middle of the board where it had started. One morning I'd move the thumbtack an inch in this direction, another morning an inch in that, but eventually all these movements in different directions would cancel each other out.

On the other hand – again supposing I was keeping a 26-hour day – what would happen to the thumback below the 26-hour clock? Here the answer is different. For as far as the 26-hour clock is concerned, I've been waking up at exactly the same time each "morning"! Every time I wake up, the 26-hour clock is pointing the same direction as it was the last time I woke up. So I'll keep moving the thumbtack one more inch in the same direction, until it's not even on the posterboard at all!

It follows, then, that just by seeing which thumbtack traveled the farthest from its starting point, you could figure out what sort of schedule I was on. In other words, you could infer the "period" of the periodic sequence that is my life.

And that, basically, is the quantum Fourier transform. Well, a little more precisely, the QFT is a *linear transformation* (indeed a unitary transformation) that maps one vector of complex numbers to another vector of complex numbers. The input vector has a nonzero entry corresponding to every time when I wake up, and zero entries everywhere else. The output vector records the positions of the thumbtacks on the posterboards (which one can think of as points on the complex plane). So what we get, in the end, is a linear transformation that maps a quantum state encoding a periodic sequence, to a quantum state encoding the *period* of that sequence.

Another way to think about this is in terms of *interference*. I mean, the key point about quantum mechanics – the thing that makes it different from classical probability theory – is that, whereas probabilities are always non-negative, *amplitudes* in quantum mechanics can be positive, negative, or even complex. And because of this, the amplitudes corresponding to different ways of getting a particular answer can "interfere destructively" and cancel each other out.

And that's exactly what's going on in Shor's algorithm. Every "parallel universe" corresponding to an element of the sequence contributes *some* amplitude to every "parallel universe" corresponding to a possible period of the sequence. The catch is that, for all periods other than the "true" one, these contributions point in different directions and therefore cancel each other out. Only for the "true" period do the contributions from different universes all point in the *same* direction. And

that's why, when we measure at the end, we'll find the true period with high probability.

Questions for next time:

1. Can QCs be built?
2. What are the limits of QCs?
3. Anything beyond QCs?

6.045J / 18.400J Automata, Computability, and Complexity

Spring 2011