Lecture 21, Thurs April 6: Continued Fractions, Shor Wrap-Up

Today we'll finish Shor's algorithm and then discuss some of its implications.

Last we saw our protagonists, they were in a superposition of the form

$$|r\rangle + |r+s\rangle + |r+2s\rangle + \ldots,$$

and we were trying to use the Quantum Fourier Transform (QFT) to extract the period s. Our first order of business was to give a polynomial-size quantum circuit to implement the QFT. Our second order of business was to understand what we observe after we apply the QFT and then measure in the computational basis.

Recall that there were two cases:

If s divides Q, then each possible measurement outcomes has either perfectly constructive or perfectly destructive interference. And the outcomes with constructive interference---i.e., the ones that we could see---are all integer multiples of Q/s. From a few random such outcomes, it's easy to recover s itself, given our knowledge of Q.

If s doesn't divide Q, then the pattern of constructive and destructive interference is no longer perfect, but has some noise to it.

That's because $k \frac{Q}{s}$ is no longer generally an integer, but the algorithm's output still needs to be an integer. So we effectively get a rounding effect, where the nearest integers to $k \frac{Q}{s}$ have the strongest constructive interference.

So we run the algorithm once and get an integer, let's say $l_1 = \lfloor k \frac{Q}{s} \rceil$, then run it again to get l_2, l_3, etc. And the question is then: given these integers, almost all of which are close to integer multiples of Q/s, how do we use them to deduce s itself?

This brings us to the final step of Shor's algorithm, which is another piece of classical number theory called the…

The Continued Fraction Algorithm

Whenever an outcome j is observed, we'd like to determine whether it's close to an integer multiple of Q/s, and if so what the multiple is. This is where we use continued fractions.

It's easiest to illustrate with an example. Let's look at the continued fraction expansion of an approximation of π, 3.14.

$$3.14 = 3 + \frac{14}{100} = 3 + \frac{1}{\frac{100}{14}} = 3 + \frac{1}{7 + \frac{2}{14}}$$

The idea is that we keep pulling out the largest integer we can and rewriting, until we have an approximation of Q/j to within an accuracy of about $1/Q^2$.

The reason why the method works is that s is a relatively small integer, so $\frac{Q}{s}$ is not only rational but has a relatively small denominator.

In more detail, let's write $l = k\frac{Q}{s} \pm \varepsilon$, where ε is some small value.

Then we divide the above equation through by Q, to get $\frac{l}{Q} = \frac{k}{s} \pm \frac{\varepsilon}{Q}$.

And so… $\left|\frac{l}{Q} - \frac{k}{s}\right| \leq \frac{\varepsilon}{Q}$.

We'll exploit the key inequality above, along with:

- the fact that we know l
- the fact that we know Q (because we picked it: it's at most $\sim N^2$)
- the fact that we know that s isn't too large

$s \leq N$ because the order of the multiplicative group is less than N, and the order of any element in the group is at most the number of elements.

So $\frac{l}{Q}$ isn't just close to *any* rational number $\frac{k}{s}$, it's close to a rational number with a pretty small denominator, and <u>that</u> doesn't happen by random chance.

There's math that backs this up.

This is the reason why we set Q to be $\sim N^2$ in the first place. Doing so ensures that the rational approximation $\frac{k}{s}$ to $\frac{l}{Q}$ is more-or-less unique, and moreover that there's an efficient algorithm to find it.

Suppose I give you a rational number, say 0.25001, and I tell you that it's close to a rational number with an unusually small denominator. How could you figure out which such rational number it's close to, without having to try *all possible* small denominators, of which there might still be too many?

In this particular example, you just stare at the thing, and immediately see ¼ is the answer! OK, but what would be a more systematic way of doing it?

The more systematic way is to expand the input number as a continued fraction, until the leftover part is so small that we can safely discard it. To illustrate:

$$\frac{25001}{100000} = \frac{1}{\frac{100000}{25001}} = \frac{1}{3 + \frac{24997}{25001}} = \frac{1}{3 + \frac{1}{\frac{25001}{24997}}} = \frac{1}{3 + \frac{1}{1 + \frac{4}{24997}}}$$

Now we've reached $\frac{4}{24997}$, a number small enough for us to discard, which leaves us with

$$\sim \frac{1}{3 + \frac{1}{1}} = \frac{1}{4}$$

So now we have a way to find $\frac{k}{s}$. <u>Are we done?</u>

Well, we still have the same difficulty that we encountered in the s divides Q case, namely that k and s might share a nontrivial divisor. If, for example, k and s were even, then we'd have no possible way to tell $\frac{k}{s}$ apart from $\frac{k/2}{s/2}$.

We solve this using exactly the same approach as before: we repeat the algorithm several times to generate $\frac{k_1}{s_1}$, $\frac{k_2}{s_2}$, $\frac{k_3}{s_3}$, etc.

One can then show that the least common multiple of the s_i's will be s itself, with a sufficiently high probability.

Today, half of pop-science articles *still* say that "quantum computers would factor numbers by trying all the possible divisors in parallel."
If you've taken anything away from our discussion of how Shor's algorithm works, I hope you now agree that it's more subtle than that!

In the next lecture, we'll see how a quantum speedup for "pure, brute-force search" *does* exist, but it's not exponential, but "merely" quadratic.

The ink wasn't dry on Shor's paper before people started asking…
What else might Shor's algorithm be good for, besides factoring?

For starters, as we mentioned a couple lectures ago, and as Shor showed in his original paper, it also gives exponential speedup for **Discrete Log**:

Given a prime p, and an integer g such that $g^x = a \pmod{p}$. Find x.

This is how Shor's algorithm breaks the Diffie-Hellman cryptosystem.

And it was noted shortly afterward that Shor's algorithm can also be modified to break **Elliptic Curve Cryptosystems**. Indeed, people quickly figured out that Shor's algorithm can be modified to solve pretty much *any* problem related to finding hidden structures in abelian groups.

Almost all the public-key cryptosystems that we currently use involve finding such hidden structures.

In the years after Shor's algorithm, a lot of research in quantum algorithms was directed towards answering the question:

"To what extent can we generalize Shor's algorithm to solve problems about *non*-abelian groups?"

By now, though, many people have given up on this direction. It's very, very hard.

Why did people care about non-abelian groups? Well, if Shor's algorithm could be generalized to handle them, there are two famous problems that would help us solve.

1. **Graph Isomorphism**

This is where you're given two graphs, and need to decide whether they're isomorphic. It's a problem that no one yet knows how to solve in polynomial time, but that famously seems to have "too much structure" to be NP-complete.

In the early 1970s, when Leonid Levin co-discovered the theory of NP-completeness, legend has it that he sat on his discovery for more than a year because he was trying to show that Graph Isomorphism is NP-complete---something that we now believe is impossible.

People quickly realized that if you could generalize Simon's and Shor's algorithms to a situation where the underlying group is the symmetric group S_n, instead of an abelian group like $\mathbb{Z}_2^n$ or $\mathbb{Z}_N^\times$, then it would solve Graph Isomorphism in quantum polynomial time.

In 2016, though, Babai (who's been studying Graph Isomorphism for forty years) found a classical algorithm to solve Graph Isomorphism in *quasipolynomial* time, meaning $n^{\text{polylog}(n)}$.

Many people suspect that Graph Isomorphism is in P, for one thing because the problem is easy in practice almost all of the time. In any case, since we now know that Graph Isomorphism is at worst quasipolynomial classically, there's no longer any possibility of getting an *exponential* quantum speedup for the problem.

2. Lattice-Based Cryptography

There's a set of public-key cryptosystems based on lattices, which are becoming increasingly important theoretically and even practically, and we don't know how to break (yet) even with a quantum computer.

Given a collection $\{z_1,...,z_n\}$ of vectors in $\mathbb{R}^n$, the lattice spanned by the collection is the set of all integer linear combinations of the vectors:

$$L = \{a_1 z_1 + \ldots + a_n z_n : a_1, \ldots, a_n \in \mathbb{Z}\}$$

A typical problem would be: "given $\{z_1,...,z_n\}$, find the shortest nonzero vector in L---or at least, a vector that's within a $\sqrt{n}$ factor of being the shortest."

It turns out that you can create entire public-key cryptosystems around these sorts of problems.

There was an important result by **Regev (2005)**, which says that we could break lattice-based cryptosystems if we could generalize Shor's algorithm to work for a nonabelian group called the dihedral group.

Needless to say---because otherwise I would've told you!---no one has yet succeeded in doing so.

So, lattice-based crypto is an attractive alternative to RSA and Diffie-Hellman for those who are paranoid about quantum computers. But it's also attractive for other reasons, including the prospect of **Fully Homomorphic Encryption**: the ability to do arbitrary computations on encrypted data without ever decrypting it. This would let people submit their data to cloud computing servers, and then get back the results, without the cloud server ever learning what computation it did. In 2009 Craig Gentry proposed the first Fully Homomorphic Encryption scheme using lattice-based crypto; since then other schemes have been proposed. These are not schemes that we know how to break even using a quantum computer.

There's still a practical problem with these schemes: the key sizes, message sizes, and computation times tend to be huge. But the schemes have been steadily improving, and many of them are now either practical or nearing practicality.