CSE439 Fall 2024 Week 9: Shor's Algorithm

In general, a **period** of a function f is a value r such that for all x,

$$f(x+r) = f(x).$$

The string s of the "promise property" in Simon's algorithm actually obeys this definition, even though it is a vector not a scalar. When Peter Shor read Simon's paper, he conceptualized that the final Hadamard transform *amplified* the periodic structure in the form of peaks and troughs of waves. The "trough" is how having $a \bullet s = 1$ made the two terms in the amplitude cancel, whereas having $a \bullet s = 0$ made them add with the same sign and hence concentrate the resulting probabilities on those cases.

Now, ahem, converting *periodic* structure into *peaks* is really the job of the *Fourier transform*, not the Hadamard transform. And the Fourier transform does this with numeric data, not just binary-string data. Shor conceptualized that replacing the final Hadamard transform with the **quantum Fourier transform** (**QFT**) might allow a similar concentration that makes a numeric period r emerge. And there is one such function and period of pre-eminent interest in cryptography... Incidentally, the QFT on n qubits is just the same as the ordinary Discrete Fourier Transform (DFT) on vectors of length $N = 2^n$. The circumstance that the QFT can be applied with $O\left(n^2\right)$ quantum effort---so the theory of quantum circuits tells us---is what makes the difference.

Periodic Functions

The important example of a periodic function is **modular exponentiation**:

$$f_a(x) = a^x \bmod M.$$

Here a is a number in $\{0, 1, \ldots, M-1\}$ that is **relatively prime** to M. This means that a does not share a prime divisor with M. When $M = pq$ is the product of two different primes p and q, this simply means that a is not divisible by p or by q. If a and M did share a divisor p, then a^x would always be a multiple of p, and $a^x \bmod M$ is also a multiple of p because p divides M too. So you would not get all of the possible values modulo M. When a is relatively prime to M, what you always get is a number relatively prime to M. This is worth spelling out more than the text does:

Definition: $G_M = \{1\} \cup \{a : 1 < a < M \textit{ and a is relatively prime to } M\}$.

Theorem: G_M forms a **group** under multiplication.

A **group** is a set G with a distinguished element 1 together with an operation $\odot$ that satisfies the following axioms:

- For all $g \in G$, $g \odot 1 \;=\; 1 \odot g \;=\; g$.
- For all $g \in G$ there is a unique $h \in G$ such that $gh = 1$ and $hg = 1$. We write $h = g^{-1}$.

For example, the $n \times n$ unitary matrices U form a group with $U^{-1} = U^*$. Well, the numbers in modular arithmetic form groups that are simpler to understand.

When $M = pq$ is a product of two primes, the size of G_M is exactly $(p-1)(q-1)$. (The general name for the size of G_M is the **totient** function of M, devised by and often named for the mathematician Leonhard Euler.) The consequence of G_M being a group that we need is:

Corollary: For all $a \in G_M$ there is a positive integer r such that $a^r \equiv 1 \bmod M$.

The least such r is exactly the period of $f_a(x)$ that we want to find. It always divides $|G_M|$, so when $M = pq$ we get that r divides $(p-1)(q-1)$. You might think this should narrow down the possibilities, but:

- We don't actually get the value $m = (p-1)(q-1)$ factored for us---we don't even know m because we don't know how to factor $M \;=:\; pq$ to begin with.
- Compared to the number n of bits or digits of M, which is the complexity parameter we care about, the range of numbers less than m we might have to check is exponential in n.
- By the way, the number x in a^x can be exponential in n, so it looks like it takes too long to compute $f_a(x)$ to begin with. However, by **iterated squaring modulo** M we can compute the following values in $\widetilde{O}(n^2)$ time: $a_1 \;=\; a^2 \bmod M$, $a_2 \;=\; a_2^2 \bmod M \;=\; a^4 \bmod M$, $a_3 \;=\; a_2^2 \bmod M \;=\; a^8 \bmod M$, $a_4 \;=\; a_3^2 \bmod M \;=\; a^{16} \bmod M$, and so on up to $a_{n-1} \;=\; a_{n-2}^2 \bmod M \;=\; a^{n-1} \bmod M$. Then we need only multiply together those a_i such that x as a binary number includes 2^i. This needs only $2n$ multiplications and mod-M reductions of n-bit numbers, so it is doable in $\widetilde{O}(n^2)$ time using an $\widetilde{O}(n)$-time integer multiplication algorithm. (Or we can say $O(n^3)$ time using the simple multiplication algorithm. The **RSA cryptosystem** uses modular exponentiation too---and this time is largely why your credit card needed a chip.)

Nevertheless, if we *do* find the period r---for a "good" value a which we stand a fine chance of picking at random from G_M---then it was known long before Peter Shor found his algorithm in 1993 that we can go on to find p and q by classical efficient means.

Theorem: There is a classical randomized algorithm that, when provided a ***function oracle*** $g(M, a) \;=\;$ some integer multiple of the period of $f_a \bmod M$, finds a factor of M in expected polynomial time. That is, **Factoring** is in BPP^g.

The proof is the entire content of Chapter 12. Lipton and I bundled this up into a separate chapter so that instructors would have the freedom to skip it, as we'll do for the time being. (2024: It will be in a replacement lecture done online via Zoom.) So we can focus on the task of finding r (or at least a multiple of r) via *quantum means*.

Shor's Theorem: **Factoring** is in BQP.

Steps of Shor's Algorithm

1. Given M, use classical randomness to guess a number a between 2 and $M-1$.
2. Use Euclid's algorithm to find $\gcd(a,M)$. If it gives a number $c > 1$, then "ka-ching!"---we got a divisor of M. Since both c and M/c are below $M/2$, we can recursively factor both of them.
3. If it gives $\gcd(a,M) = 1$, then we know $a \in G_M$. In the important $M = pq$ case, this had probability $\frac{(p-1)(q-1)}{pq}$ and so was pretty likely anyway. By the way, Euclid's algorithm also gives you a number b such that $ab = 1 \bmod M$. But it doesn't give you this b as a power of a (to wit, as $b = a^{r-1} \bmod M$), which is what you'd need to get r.
4. To give some slack, we choose a number $Q = 2^\ell \approx M^2$ and expand the domain of $f_a(x)$ to include x in the interval up to $Q-1$, not just up to $M-1$. The range is still 1 to $M-1$. So our domain is x in the range 0 to $2^\ell - 1$, which uses $\ell \approx 2n$ bits. This gives us quadratically many "ripples" of the period, which in turn helps the trigonometric analysis in the body of the proof.
5. The quantum circuit begins with q-many Hadamard gates, followed by a quantum implementation of the $n^{O(1)}$ classical gates needed to compute modular exponentiation. This produces the functionally superposed quantum state
$$\Phi_f = \frac{1}{\sqrt{N}} \sum_{x \in \{0,1\}^\ell} |x f_a(x)\rangle.$$
6. Apply the QFT (or its inverse) to the first ℓ qubits.
7. Then *measure* the whole result. Curiously, we ignore what happens in the "$f_a(x)$" portion of the circuit. The fact that those final n qubits were entangled with the first ℓ qubits is enough. So we let our output w in the "x-space" be the first ℓ bits of the measured result over the binary standard basis.

My own quantum circuit simulator draws an ASCII picture of the Shor circuit, here for $M = 21 = 3*7$ (where I guessed $a = 5$), which gave $\ell = 9$ since $2^9 = 512$ is the next power of 2 after $M^2 = 441$:

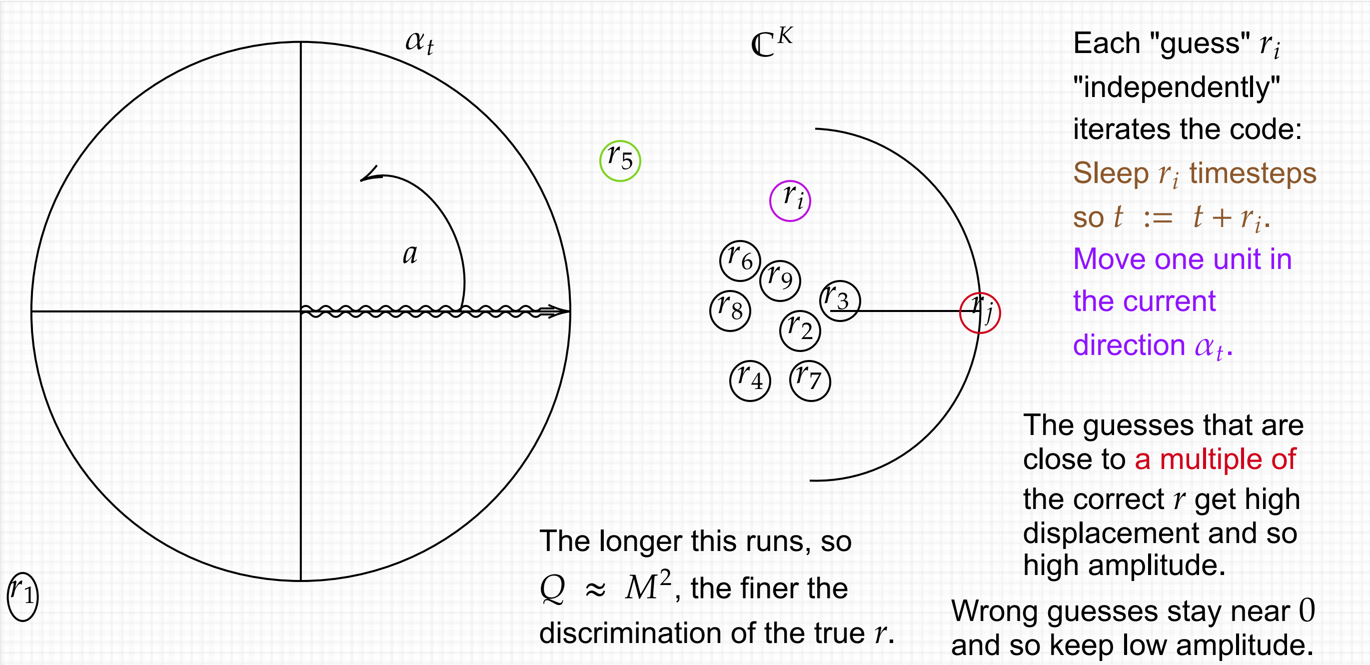
```
->--[H]--[MODEXP]--[QFTI]--<?
             |        |
->--[H]------|-----[QFTI]--<?
             |        |
->--[H]------|-----[QFTI]--<?
             |        |
->--[H]------|-----[QFTI]--<?
             |        |
->--[H]------|-----[QFTI]--<?
             |        |
->--[H]------|-----[QFTI]--<?
             |        |
->--[H]------|-----[QFTI]--<?
             |        |
->--[H]------|-----[QFTI]--<?
             |        |
->--[H]------|-----[QFTI]--<?
             |
->-------[MODEXP]----------<?
             |
->-----------|-------------<?
             |
->-----------|-------------<?
             |
->-----------|-------------<?
             |
->-----------|-------------<?
```

But there isn't any more to the quantum circuitry than that. It's all simply: compute a giant functional superposition and apply QFT (or its inverse) to it.

The analysis establishes that with pretty good probability already in one shot, the output y reveals the period r by a followup classical means. And with initial good probability over the choice of a, the resulting value r unlocks the key to factoring M. We will focus on understanding why the measured y has much to do with the period r to begin with. Then basic point---which has been known for centuries---is that the Fourier transform converts *periodic data* to *peaked data*. Here is how the simple quantum circuit above applies this fact.

The Intuition (See also Scott Aaronson, https://www.scottaaronson.com/blog/?p=208)

Let r stand for the true period of f. Let a be any element of the group G_M of size $(p-1)(q-1)$. Then we will picture a as a "crazy clock" that jumps a units *counter*-clockwise at each time step.



With fairly high probability, measurement---followed by figuring needed to get the guessed r_i from the measurement---yields a multiple of r. The true r is the least of the multiples. It is individually the most likely value returned and is also returned with reasonable probability. A non-least r might work anwyay. We can tell whether r works by seeing if the classical part gives us p or q, else we just try the quantum process again.

Heading into the analysis, however, we need to say exactly what the measured string w actually represents. In general, the angle α represented by a (when we actually use the complex plane to model the "crazy clock") will not be a whole-number fraction of the circle. But let us first suppose it is. Then the smallest period r (i.e., the true period) will go exactly once around the circle and back to angle α as represented by a. So suppose r_i is a correct guess of r. Then with high probability, the output w of the measurement has the same angle α. Since angles add when we multiply complex numbers, this means $r\alpha$ takes us once around the circle. This in turn means that α is the *reciprocal* of r with regard to the circle. So w would be close to this reciprocal.

In the general case, we have to go some number t times around the circle before we get exactly back to a. That is, we have $r\alpha = t$ with respect to the circle. So $\alpha = \frac{t}{r}$ times whatever number Q represents the extent of once-around-the-circle in the units we are using. This finally means that w should be close to $\frac{tQ}{r}$ in these units. The w needs to be close enough to pull one final switcharoo: We don't know what t is either, but from $w \approx \frac{tQ}{r}$ we get $r \approx t\frac{Q}{w}$. Since r has to be an integer, we just need to find a t that multiply the fraction $\frac{Q}{w}$ into being real close to an integer. It turns out this will work when the additive error in the measured w relative to the "true amplifying direction" $\frac{tQ}{r}$ is at most ± 0.5 in the circle's units. Choosing Q high enough makes those units fine enough for this to work. The "analysis of the quantum part" tells how often the measured w is close enough to be "good." (As was the case with Simon's algorithm, the text re-uses the letter "x" to denote the particular string from the "x-space" that was obtained in the measurement.)

Simulation Interlude

Before we go to this analysis, let's see a brute-force simulation of Shor's algorithm. It pretty much builds the concrete "mazes" for $\ell + n$ qubits and simulates all the legal "Feynman mouse paths" through them. The run of my simulator on $M = 21$ and $a = 5$ succeeded on the second try:

```
About to do try 1 of sampling QFT applied to 1010101011010010100 with status now PROBS_ENUMERA
Sampling with status PROBS_ENUMERATED:
Base probability for conditionals: 0.166015625000
Current: 0 with probability 0.083007813 on rolling 0.325191374; last 0 prob = 0.500000000
Current: 00 with probability 0.055282593 on rolling 0.563273639; last 0 prob = 0.665992647
Current: 001 with probability 0.027659269 on rolling 0.559076137; last 0 prob = 0.499674899
Current: 0010 with probability 0.027418884 on rolling 0.941772811; last 0 prob = 0.991309060
Current: 00101 with probability 0.027183985 on rolling 0.139894580; last 0 prob = 0.008567052
Current: 001010 with probability 0.026380861 on rolling 0.938149097; last 0 prob = 0.970455980
Current: 0010101 with probability 0.025648040 on rolling 0.595421001; last 0 prob = 0.02777850
Current: 00101010 with probability 0.020074378 on rolling 0.114898273; last 0 prob = 0.7826866
Current: 001010101 with probability 0.018908726 on rolling 0.791199151; last 0 prob = 0.058066
sampled output vector: 00101010110100
time cost: 1.23308 milliseconds.

Measured 001010101 as 85 giving 0.166015625
Fractional approximation is 1/6
; Possible period is 6
; Unable to determine factors, we'll try again.
Let's take a free random crack at it without the QFT application...
Fractional approximation is 2/3
; Odd denominator, trying to expand by 2.
; Possible period is 6
; Unable to determine factors, we'll try again.

About to do try 2 of sampling QFT applied to 1010101011010010100 with status now PROBS_ENUMERA
Sampling with status PROBS_ENUMERATED:
Base probability for conditionals: 0.166015625000
Current: 1 with probability 0.083007813 on rolling 0.527169932; last 0 prob = 0.500000000
Current: 10 with probability 0.055282593 on rolling 0.051374227; last 0 prob = 0.665992647
Current: 100 with probability 0.027623324 on rolling 0.277237177; last 0 prob = 0.499674899
Current: 1000 with probability 0.027576410 on rolling 0.189192738; last 0 prob = 0.998301645
Current: 10000 with probability 0.027567765 on rolling 0.562397971; last 0 prob = 0.999686499
Current: 100000 with probability 0.027564179 on rolling 0.523783427; last 0 prob = 0.999869929
Current: 1000000 with probability 0.027562462 on rolling 0.694951445; last 0 prob = 0.99993770
Current: 10000000 with probability 0.027561612 on rolling 0.646817553; last 0 prob = 0.9999691
Current: 100000000 with probability 0.027561188 on rolling 0.353241189; last 0 prob = 0.999984
sampled output vector: 10000000010100
time cost: 1.2329 milliseconds.

Measured 100000000 as 256 giving 0.500000000
Fractional approximation is 1/2
; Possible period is 2
; Success: 21 = 3 * 7
Success after 2 xy sample(s) plus 2 QFT sample(s).
```

The detailed analysis from chapter 11 (continuing into chapter 12) will come in week 10.