Chapter 7

Period Finding and Factoring

Let's say we are given a black box for computing a periodic function, i.e. a function where

$$f(x) = f(y) \text{ if and only if } x \equiv y \pmod{r}$$

The goal of a period finding algorithm is to find r.

The algorithm for period finding is very similar to Simon's algorithm, in fact we can think of it as a generalization of Simon's algorithm. The steps we follow are very similar.

Classically, we could solve this problem by querying our function with subsequent inputs until the function repeats. This takes $O(r) = O(2^n)$ queries to the function. There are other ways to solve this problem, but it can be shown that all classical algorithms solve this problem in exponential time.

With a *Quantum* computer, we can access the function in *superposition* to query the function with $N = 2^n$ inputs for each n qubits at the same time. The key ingredients to our approach will be the period/wavelength and linear shift properties of the Fourier transform. We first access the function in superposition to create a periodic superposition that may not start from 0 (it includes alinear shift), and then take its Fourier transform to get rid of the linear shift.

This approach is very similar to the approach of Simon's algorithm, and is the historical motivation for the period finding algorithm. Lets examine the details.

Step 1: Prepare the periodic superposition $\sqrt{\frac{r}{N}} \sum_{j=0}^{N/r-1} |x_0 + jr\rangle$

Step 2: Fourier *sample* to measure some $y = \frac{kN}{r}$ for $k \in \{0, 1, \cdots, r-1\}$.

Step 3: Repeat until there are enough such y's so that we can compute their greatest common divisor and solve for r.

Step 1. It is best to start a quantum algorithm with the easily prepared state $|0\rangle$, but we want to access our function in superposition: we need the state $\frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle |0\rangle$. To prepare this state, we

just implement QFT_N on the first n[1] qubits:

$$|0\rangle|0\rangle \xrightarrow{QFT_N} \frac{1}{\sqrt{N}}\sum_{x=0}^{N-1}|x\rangle|0\rangle$$

Next, as in Simon's algorithm, we access our function in superposition. Let U_f be the unitary transformation that carries out our function, and implement it:

$$\frac{1}{\sqrt{N}}\sum_{x=0}^{N-1}|x\rangle|0\rangle \xrightarrow{U_f} \frac{1}{\sqrt{N}}\sum_{x=0}^{N-1}|x\rangle|f(x)\rangle$$

To get a periodic superposition out of this, we measure $|f\rangle$. Then $|f\rangle$ must collapse into some value $f(x_0)$. Furthermore, because measuring $|f\rangle$ reveals information about $|x\rangle$, the state $|x\rangle$ will also collapse into the pre-image of $f(x_0)$. But because f is periodic, the pre-image of $f(x_0)$ is $\{x_0,\ x_0+r,\ x_0+2r,\ \cdots,\ x_0+(\frac{N}{r}-1)r\}$.

$$\frac{1}{\sqrt{N}}\sum_{x=0}^{N-1}|x\rangle\otimes|f(x)\rangle \xrightarrow{\text{measure}|f\rangle} \sqrt{\frac{r}{N}}\sum_{i=0}^{N/r-1}|x_0+ir\rangle|f(x_0)\rangle$$

Now our first register is in a periodic superposition, where the period is the same as the period of the function! But we can't just measure, because each time we run the algorithm, we might measure a different value of $|f\rangle$, thus obtaining a periodic superposition that is linearly shifted by some other x_0

Step 2: We can't just measure our superposition right away, because that would destroy the superposition. And because of the random linear shift x_0, a measurement wouldn't reveal any useful information. Instead, we will rely on the properties of the Fourier transform to retrieve the information we want. Remember that if f is periodic with period r, then $\hat{f}$ is periodic with period N/r. Furthermore, remember that we only see the effect of the linear shift x_0 in the phase of $\hat{f}$. Therefore if we take the Fourier transform of the first register, we will be left only with states that are multiples of N/r.

$$\sqrt{\frac{r}{N}}\sum_{i=0}^{N/r-1}|ir+x_0\rangle \xrightarrow{QFT_N} \frac{1}{\sqrt{r}}\sum_{i=0}^{r-1}\left|i\frac{N}{r}\right\rangle\phi_i$$

where ϕ_i is the (unimportant) phase associated with each term due to the linear shift x_0.

Now we can measure and retrieve $k\frac{N}{r}$ for some integer k!

Step 3 Now we repeat the algorithm to retrieve several distinct multiples of N/r. Once we have enough values, we can compute their GCD to retrieve N/r. N is a given in the problem, so it is easy to compute r. Computing GCD is easy thanks to Euclid's algorithm.

[1] Don't let the different n's confuse you. If there are n qubits, then we need $N = 2^n$ complex numbers to describe the system.

How long should we expect this to take? Let us compute the chance of finding the correct period after t samples. Suppose after finding t distinct multiples of N/r, we have not found the desired period N/r, but instead an integer multiple of it, say $\lambda N/r$. This means that each of the t samples must be a multiple of $\lambda N/r$. There are exactly $N/(\lambda N/r) = r/\lambda$ such multiples of $\lambda N/r$. And since there are r multiples in total, the probability of measuring a multiple of $\lambda N/r$ is $1/\lambda$. Therefore,

$$\Pr[\text{gcd is a multiple of } N/r] = \left(\frac{1}{\lambda}\right)^t \leq \left(\frac{1}{2}\right)^t,$$

and we err with probability

$$\Pr[\text{gcd} > N/r \text{ after } t \text{ samples}] \leq N\left(\frac{1}{2}\right)^t.$$

Therefore we must repeat the period finiding circuit $O(\log N)$ times to be confident in our solution.

The above algorithm can be summed up by Figure 1:

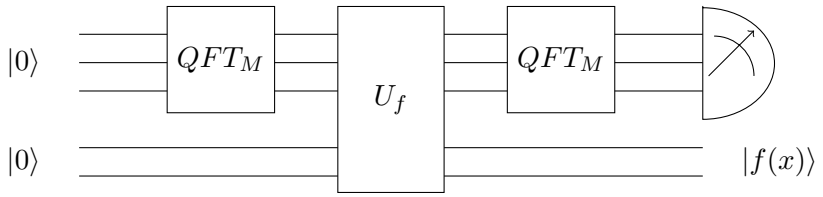

Figure 7.1: Circuit for period finding

Example

In this example we find the period of the function $f(x) = x \pmod 2$. It is easy to see that the period of this function is $r = 2$

We will use a 3-qubit system so that $N = 8$. It is a good rule of thumb to choose $N \gg r$. The first step is to apply the quantum Fourier transform:

$$|0\rangle|0\rangle \xrightarrow{QFT_8} \frac{1}{\sqrt{8}}\sum_{x=0}^{7}|x\rangle|0\rangle$$

Next we apply our function.

$$\frac{1}{\sqrt{8}}\sum_{x=0}^{7}|x\rangle|0\rangle \xrightarrow{U_f} \frac{1}{\sqrt{8}}\sum_{x=0}^{7}|x\rangle|x \bmod 2\rangle$$

The next step is to measure $|f\rangle$. Then $|f\rangle$ must collapse into either $|0\rangle$ or $|1\rangle$. For the purpose of demonstration, lets say our measurement returns $|f(x)\rangle = |1\rangle$. Then x must be odd.

$$\frac{1}{\sqrt{8}}\sum_{x=0}^{7}|x\rangle \otimes |f(x)\rangle \xrightarrow{measure|f\rangle} \frac{1}{2}(|1\rangle + |3\rangle + |5\rangle + |7\rangle) \otimes |1\rangle$$

Now we need to extract the period of the first register without the obnoxious linear shift. So once again we apply the Fourier transform.

$$\frac{1}{2}(|1\rangle + |3\rangle + |5\rangle + |7\rangle) \xrightarrow{QFT_8} \frac{1}{\sqrt{2}}(|0\rangle - |4\rangle)$$

Note: If instead of measuring $|f\rangle = |1\rangle$ we had measured $|f\rangle = |0\rangle$, there would be a different linear shift. But the properties of Fourier transform dictate that this only effects the *phase* of the Fourier transform. In other words, that last step would have looked like $\frac{1}{2}(|0\rangle + |2\rangle + |4\rangle + |6\rangle) \xrightarrow{QFT_8} \frac{1}{\sqrt{2}}(|0\rangle + |4\rangle)$. This agrees with what we know about the principal of deferred measurement.

Finally, if we take a few measurements we will be sure to measure both $|0\rangle$ and $|4\rangle$. Therefore $N/r = 4$, and since $N = 8$, it is clear that $r = 2$.

Summary

Now that we understand how the algorithm works, we can write it without some of the fluff.

$$|0\rangle\,|0\rangle \xrightarrow{QFT_M} \frac{1}{\sqrt{M}} \sum_{x \in \mathbf{Z}_M} |x\rangle\,|0\rangle \tag{7.1}$$

$$\xrightarrow{f} \frac{1}{\sqrt{M}} \sum_{x \in \mathbf{Z}_M} |x\rangle\,|f(x)\rangle \tag{7.2}$$

$$\xrightarrow{\text{measure 2nd register}} \sqrt{\frac{r}{M}} \sum_{k=0}^{\frac{M}{r}-1} |x_0 + kr\rangle\,|f(x_0)\rangle \tag{7.3}$$

$$\xrightarrow{QFT_M} \sqrt{\frac{r}{M}} \frac{1}{\sqrt{M}} \sum_{y \in \mathbf{Z}_M} \alpha_y\,|y\rangle \tag{7.4}$$

where $\alpha_y = \sum_{k=0}^{M/r-1} \omega^{(x_0+kn)y} = \omega^{x_0 y} \sum_k \omega^{kry}$.

There are two cases for y:

1. Case 1: y is a multiple of $\frac{M}{r}$.

   In this case, then $\omega^{kry} = e^{2\pi i ry/M} = e^{n2\pi i} = 1$. So $\alpha_y = \frac{\sqrt{r}}{M}\frac{M}{r} = \frac{1}{\sqrt{r}}$. This should be thought of as *constructive interference* due to the final QFT_M.

   Note that there are r multiples of M/r. Because $\sum_1^r (\frac{1}{\sqrt{r}})^2 = 1$, we know that $\alpha_y = 0$ for any y that is *not* a multiple of $\frac{M}{r}$ by normality.

2. Case 2: y is not a multiple of $\frac{M}{r}$.

   We already know that α_y must be 0 from the previous case. Furthermore, note that $\omega^{ry}, \omega^{2ry}, \ldots$ are evenly spaced vectors in the complex plain of unit length around the origin. Summing over these vectors we see that α_y is 0. This can be viewed as *destructive interference* due to the final QFT_M.

The interference that occurs in the final step is one reason quantum computers are so well equipped for period finding. We call it interference because it is additions in the *phase* that cause the cancellations.

7.1 Shor's Quantum Factoring Algorithm

One of the most celebrated algorithms for quantum computers is Shor's Algorithm for factoring. The time it takes for a classical computer to factor some number with n digits grows exponentially with n, meaning that numbers with many digits take a very long time for a classical computer to factor. RSA cryptography and other cryptography algorithms take advantage of this difficulty, and as a result a large amount of information is protected by large semi prime numbers (products of two primes).

The time it takes a quantum computer to factor an n-digit number grows as a polynomial in n.

The reason we focused so much attention on period finding is because the problem of factoring can be reduced to the problem of period finding thanks to modular arithmetic. This isn't obvious, but with a little setup we can understand why.

Setup

In modular arithmetic, we call a number x a non-trivial square root of 1 modulo N if $x^2 \equiv 1 \pmod{N}$ and $x \neq \pm 1$. For example, 2 is a non-trivial square root of unity modulo 3 because $2^2 = 4 \equiv 1 \pmod{3}$. It turns out that if we can find such an x, we can factor N. Later we will see that we can use period finding to find x. This idea is summed up in the following lemmas.

Factoring is equivalent to finding a nontrivial squareroot of $1 \mod N$. Let $x \neq \pm 1 \mod N$ and $x^2 = 1 \mod N$. Then $x^2 - 1 = 0 \pmod{N}$ so that $x^2 - 1$ is a multiple of N. Factoring, we see that $N \mid (x+1)(x-1)$, but because $x \neq \pm 1 \pmod{N}$, $N \nmid (x \pm 1)$.

Therefore, $\gcd(N, x+1)$ and $\gcd(N, x-1)$ are factors of N, and greatest common divisor is easy to compute with Euclid's algorithm.

Example: Suppose we want to factor the number 15. It is easy to see that $4^2 = 16 \equiv 1 \mod 15$, but $4 \neq \pm 1 \mod 15$. So 4 is a non-trivial square root of unity modulo 15. Then gcd(15,5) and gcd(15,3) are factors of 15. Sure enough we see that $5 \cdot 3 = 15$.

Now, all we need to do is find this nontrivial squareroot of unity, and we can factor whatever number we need. As promised, we can do this with period finding, specifically by computing the order of a random integer.

The **order** of some integer x modulo N is the smallest integer r such that $x^r = 1 \mod N$. For example, the order of 2 modulo 3 is 2 since $2^2 \equiv 1$, the order of 3 modulo 5 is 4 since $3^2 = 9 \equiv 4$; $3^3 = 25 \equiv 2$; and $3^4 = 81 \equiv 1 \pmod{5}$. Another way to say this is that the order of x is just the period of the function $f(i) = x^i \mod N$.

Suppose $N = p \cdot q$, and $x \in \mathbf{Z}_N$, $x \neq p, q$. Then with probability $\geq 1/2$, the order s of x is even, and $x^{s/2}$ is a nontrivial square root of 1 mod N.

The proof of this statement requires results from number theory (Fermat's little Theorem, Chinese remainder Theorem) that are outside the scope of this course, so we will state it without proof. However, it should be intuitive: if you imagine the order of a number to vary randomly from one number to the next, you expect the order of a number to be even with probability about half.

Example: Find the order of 2 (mod 63), and use it to factor 63.

1. $2 = 2$
2. $2^2 = 4$
3. $2^3 = 8$
4. $2^4 = 16$
5. $2^5 = 32$
6. $2^6 = 64 \equiv 1 \pmod{63}$

so that the order of 2 is 6. Note that a quantum computer wouldn't have to compute each of these powers, it would simply use the period finding algorithm described earlier. Now we compute $2^3 = 8 \neq \pm 1$, so that $\gcd(63, 8+1) = 9$ and $\gcd(63, 8-1) = 7$ are factors of 63.

The Algorithm

When finding order using the period finding algorithm, it is important to use enough qubits. A sensible rule is that you need to use m qubits so that $2^m \gg N^2$, where N is the number we are trying to factor, because the order of a random number might be as large as N.

We now have all the necessary tools to carry out Shor's algorithm. Start by picking a random number, then use the period finding algorithm to compute its order. If the order is even, we can use it to find a nontrivial square root of unity. If the order is odd or $x^{s/2} = -1$, throw it out and start with a new number.

Because we know that the order of x will be even and $x^{s/2}$ will be a nontrivial square root with probability at least 1/2, we can be confident that we will be able to factor N in just a few runs of the algorithm. Because the time it takes to find the period grows as a polynomial in the number of bits, and the number of bits grows like $2 \log N$(by the above requirement), we expect the time it takes to factor N to grow as a polynomial in $\log N$.

Here is the circuit for Shor's Algorithm. It relies heavily on period finding, and so the circuit looks a lot like the circuit for period finding. The key difference is that we are finding the period of $f(i) = x^i$, and the number of bits we need to input is very large.

Example

Here's an example that's a little more fun. Lets factor 119. Suppose we pick the number 16 to start with.

First, we compute it's order.

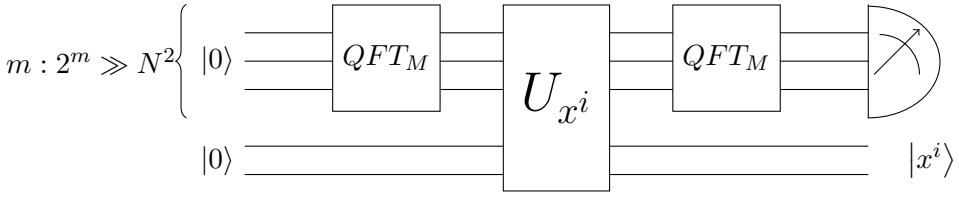

Figure 7.2: Circuit for factoring

1. $16 = 16$
2. $16 \cdot 16 = 256 \equiv 18$
3. $18 \cdot 16 = 288 \equiv 50$
4. $50 \cdot 16 = 800 \equiv 86$
5. $86 \cdot 16 = 1376 \equiv 67$
6. $67 \cdot 16 = 1072 = 119 \cdot 7 + 1 \equiv 1$

so that the order of 16 mod 119 is 6. Now, we compute $16^3 \equiv 50$. Gcd(49,119) = 7, so 7 is a factor of 119, and gcd(51, 119) = 17 which is another factor of 119.