Quantum Computing
Lecture 7

Quantum Factoring

Anuj Dawar

Quantum Factoring

A *polynomial time* quantum algorithm for factoring numbers was published by *Peter Shor* in 1994.

> *polynomial time here means that the number of gates is bounded by a polynomial in the number of bits n of the number being factored.*

The best known classical algorithms are exponential (in $n^{1/3}$).

Fast factoring would undermine public-key cryptographic systems such as RSA.

Period Finding

Suppose we are given a function $f : \mathbb{N} \to \{0, \ldots, N-1\}$ which we know is periodic, i.e.

$$f(x + r) = f(x) \quad \text{for some fixed } r \text{ and all } x.$$

Can we find the least value of r?

If we can find the period of a function efficiently, we can factor integers quickly.

Order Finding

Suppose we are given an integer N and an a with $a < N$ and

$$\gcd(a, N) = 1.$$

Consider the function $f_a : \mathbb{N} \to \{0, \ldots, N-1\}$ given by

$$f_a(x) \equiv a^x \pmod{N}$$

Then, f_a is periodic, and if we can find the period r, we can factor N.

Factoring

Suppose (for simplicity) $N = pq$, where p and q are prime. And, for some $a < N$, we know the period r of the function f_a.
Then, $a^{r+1} \equiv a \pmod{N}$, so $a^r \equiv 1 \pmod{N}$.

If r is even and $a^{r/2} + 1 \not\equiv 0 \pmod{N}$, then take $x^2 = a^r$.

$$\begin{aligned} x^2 - 1 &\equiv 0 \pmod{N} \\ (x-1)(x+1) &\equiv 0 \pmod{N} \end{aligned}$$

But,

$$\begin{array}{lcll} x - 1 & \not\equiv & 0 \pmod{N} & \text{(by minimality of } r\text{)} \\ x + 1 & \not\equiv & 0 \pmod{N} & \text{(by assumption)} \end{array}$$

Factoring–*contd.*

So, $(x-1)(x+1) = kpq$ for some k.
Now, finding $\gcd(N, x-1)$ and $\gcd(N, x+1)$ will find p and q.

If we *randomly* choose $a < N$

> *(and check that* $\gcd(a, N) = 1$*—if not, we've already found a factor of* N*)*

then, there is a probability $> \frac{1}{2}$ that

- the period of f_a is even; and
- $a^{r/2} + 1 \not\equiv 0 \pmod{N}$.

Using a Fourier Transform

A fast period-finding algorithm allows us to factor numbers quickly.

The idea is to use a *Fourier Transform* to find the period of a function f.

Note, classically, we can use the *fast Fourier transform* algorithm for this purpose, but it can be shown that this would require time $N \log N$, which is exponential in the number of bits of N.

Discrete Fourier Transform

The *discrete Fourier transform* of a sequence of complex numbers

$$x_0, \ldots, x_{M-1}$$

is another sequence of numbers

$$y_0, \ldots, y_{M-1}$$

where

$$y_k = \frac{1}{\sqrt{M}} \sum_{j=0}^{M-1} x_j e^{2\pi ijk/M}$$

or

$$y_k = \frac{1}{\sqrt{M}} \sum_{j=0}^{M-1} x_j \omega^{jk}.$$

where $\omega = e^{2\pi i/M}$.

DFT is Unitary

The *discrete Fourier transform* is a *unitary* operation on $\mathbb{C}^M$.
Writing ω for $e^{2\pi i/M}$,

$$\omega, \omega^2, \ldots, \omega^{M-1}, \omega^M = 1$$

are the Mth roots of 1.

$$D = \frac{1}{\sqrt{M}} \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{-1} \\ 1 & \omega^2 & \omega^4 & \cdots & \omega^{-2} \\ 1 & \omega^3 & \omega^6 & \cdots & \omega^{-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{M-1} & \omega^{2M-2} & \cdots & \omega \end{bmatrix}$$

Inverse DFT

The inverse of the discrete Fourier Transform is given by:

$$D^{-1} = \frac{1}{\sqrt{M}} \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^{-1} & \omega^{-2} & \cdots & \omega \\ 1 & \omega^{-2} & \omega^{-4} & \cdots & \omega^{2} \\ 1 & \omega^{-3} & \omega^{-6} & \cdots & \omega^{3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{1-M} & \omega^{2-2M} & \cdots & \omega^{M-1} \end{bmatrix}$$

Exercise: Verify that D is unitary. Verify that D^{-1} as given above is the inverse of D.

Quantum Fourier Transform

Computing the discrete Fourier transform classically takes time *polynomial in M*.

Shor showed that D can be implemented using a number of one and two-qubit gates that is only polynomial in the number of qubits

$O((\log M)^2)$.

Note: This *does not* give a fast way to compute the DFT on a quantum computer.
There is no way to extract all the complex components from the transformed state.

Fourier Transform on Binary Strings

Suppose $M = 2^n$, and let $|x\rangle$ be a computational basis state in $\mathbb{C}^M$ with binary representation $b_1 \cdots b_n$.
Let

$$\eta_j = e^{2\pi i (0 \cdot b_j b_{j+1} \cdots b_n)}.$$

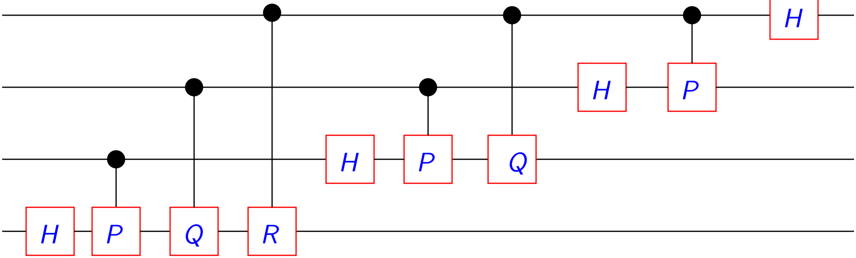
Then

$$D|x\rangle = (|0\rangle + \eta_n|1\rangle)(|0\rangle + \eta_{n-1}|1\rangle) \cdots (|0\rangle + \eta_1|1\rangle).$$

Exercise: Verify.

Quantum Fourier Transform Circuit

We can use this form to implement the *quantum Fourier transform* using Hadamard gates and conditional phase-shift gates.



In the input, the least significant bit is at the top, in the output, it is at the bottom.

Conditional Phase Shifts

Here

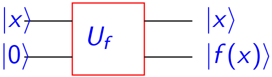
$$P = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix} \qquad Q = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix} \qquad R = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/8} \end{bmatrix}$$

Two-qubit conditional phase shift gates are actually symmetric between the two bits, despite the asymmetry in the drawn circuit.

It seems that for large n, an n-bit quantum Fourier transform circuit would require conditional phase shifts of *arbitrary precision*.
It can be shown that this can be avoided with some (but not significant) loss in the probability of success *for the factoring algorithm*.

Preparing the State

We are given an implementation of the function f as a unitary operator U_f



Where, now, each of the two input wires represents n distinct qubits

Writing $|\Psi\rangle$ for the state

$$H^{\otimes n}|0^n\rangle = \frac{1}{2^{n/2}} \sum_{x=0}^{2^n-1} |x\rangle.$$

We have,

$$U_f|\Psi\rangle|0^n\rangle = \frac{1}{2^{n/2}} \sum_{x=0}^{2^n-1} |x\rangle|f(x)\rangle.$$

First Measurement

We measure the second n qubits of the state $U_f|\Psi\rangle|0^n\rangle$ and get a value f_0. The state after measurement is:

$$(\frac{1}{\sqrt{m}}\sum_{k=0}^{m-1}|x_0+kr\rangle)|f_0\rangle.$$

where:

x_0 is the least value such that $f(x_0)=f_0$

r is the period of the function f

$m=\lfloor\frac{2^n}{r}\rfloor$.

Applying the QFT

We apply the n-qubit quantum Fourier transform to the first n bits of the transformed state.

$$
\begin{aligned}
& D\Big(\tfrac{1}{\sqrt{m}}\sum_{k=0}^{m-1}|x_0+kr\rangle\Big) \\
&\quad = \tfrac{1}{2^{n/2}}\sum_{y=0}^{2^n-1}\frac{1}{\sqrt{m}}\sum_{k=0}^{m-1}\omega^{(x_0+kr)y}|y\rangle \\
&\quad = \sum_{y=0}^{2^n-1}\omega^{x_0 y}\frac{1}{2^{n/2}\sqrt{m}}\Big(\sum_{k=0}^{m-1}\omega^{kry}\Big)|y\rangle.
\end{aligned}
$$

where $\omega = e^{2\pi i/2^n}$.

Second Measurement

The probability of observing a given state $|y\rangle$ is:

$$\frac{1}{2^n m}\left|\sum_{k=0}^{m-1}\omega^{kry}\right|^2 .$$

This probability function has peaks when $ry/2^n$ is close to an integer. Indeed, if $ry/2^n$ *is* an integer, then with probability 1 we measure a y that is a multiple of $r/2^n$.

Given an integer multiple of $2^n/r$, it is not difficult to find r.

Exponentiation

To complete the factoring algorithm, we need to check that we can also implement the unitary transform U_f for the particular function

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

with a number of quantum gates that is polynomial in $\log N$.

This is achieved through repeated squaring.

Some Points to Note

The two measurement steps can be combined at the end, with the Fourier transform applied before the measurement of $f(x)$.

The probability of successfully finding the period in any run of the algorithm is only about 0.4.
However, this means a small number of repetitions will suffice to find the period with high probability.

Putting a *lower bound* on the conditional phase shift we are allowed to perform affects the probability of success, but not the rest of the algorithm.