Quantum Computing
Lecture 7

Quantum Factoring

Maris Ozols

Quantum factoring

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

Polynomial time means that the number of gates is bounded by a polynomial in $\log N$, where $\log N$ is the number of bits required to represent the number N being factored.

The best known classical algorithm takes sub-exponential time (it is exponential in $(\log N)^{1/3}$).

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

Example

RSA-768
It has 232 decimal digits and was factored over the span of 2 years:

```
1230186684530117755130494958384962720772853569595334792197322452151726400507263657518745202199786469389956474942774063845925192557326303453731548268507917026122142913461670429214311602221240479274737794080665351419597459856902143413
=
334780716989568987860441698482126908177047949837137685689124313889828837938780022876147116525317430877378144679994 89
×
3674604366679959042824463379962795263227915816434308764267603228381573966651127923337341714339681027009279873630 8917
```

The total CPU time spent on a parallel computer amounted to approximately 2000 years on a single-core 2.2 GHz computer.

Order finding

Suppose we are given $a, N \in \mathbb{N}$ such that $a < N$ and

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

Consider the infinite sequence

$$a^0, a^1, a^2, a^3, \ldots \pmod{N}$$

Since each $a^k \in \{0, \ldots, N-1\}$, the sequence starts to repeat at some point. In particular, $a^r \equiv 1 \pmod{N}$ for some integer $r \geq 1$ since $\gcd(a, N) = 1$ (see Euler's theorem or the extended Euclidean algorithm). The order of a is the smallest such r (it is also the period of the above sequence).

Strategy: Reduce factoring to order (period) finding. We want to show that if we can find the period r of a then we can factor N.

Using order finding to factor

Assume $N = pq$, where p and q are odd primes (the general case can be handled with a little more effort).

Also, assume we have a subroutine for finding order modulo N.

Reduction:

1. Pick a random $a \in \{2, \ldots, N-1\}$ and compute $g = \gcd(a, N)$.
2. If $g \neq 1$, it is a non-trivial factor of N, so we output g and N/g and we are done. Otherwise, $\gcd(a, N) = 1$ and we continue.
3. Use the order finding subroutine to find the order r of a modulo N.
4. If r is even, let $x = a^{r/2}$ (otherwise, abort and return to 1).
5. If $x + 1 \not\equiv 0 \pmod{N}$, output $\gcd(N, x+1)$ and $\gcd(N, x-1)$ (otherwise, abort and return to 1).

Analysis of halting

Does this procedure halt? We could keep aborting in steps 4 or 5...

Fact: If N is a product of two odd primes and we choose a random $a \in \{2, \ldots, N-1\}$ such that $\gcd(a, N) = 1$, then with probability $> \frac{1}{2}$

(i) the order r of a is even and

(ii) $a^{r/2} + 1 \not\equiv 0 \pmod{N}$

In other words, in each run we abort with probability $< 1/2$. The probability that we still haven't succeeded in k rounds is thus $< 2^{-k}$.

Assume we made it to step 5 and output $\gcd(N, x+1)$ and $\gcd(N, x-1)$. Why are they factors of N?

Recovering factors from a and r

Let $N = pq$, where p and q are odd primes, and assume we have guessed a such that (i) r is even and (ii) $a^{r/2} + 1 \not\equiv 0 \pmod{N}$.

Let $x = a^{r/2}$. Since $a^r \equiv 1 \pmod{N}$, we have $x^2 - 1 \equiv 0 \pmod{N}$ so

$$(x-1)(x+1) \equiv 0 \pmod{N} \qquad (*)$$

But note that

$$x - 1 \not\equiv 0 \pmod{N} \quad \text{(by minimality of } r\text{)}$$
$$x + 1 \not\equiv 0 \pmod{N} \quad \text{(by assumption)}$$

The condition $(*)$ is equivalent to:

$$(x-1)(x+1) = kpq \qquad \text{for some integer } k$$

Since neither $x - 1$ nor $x + 1$ is a multiple of N, computing $\gcd(N, x-1)$ and $\gcd(N, x+1)$ will find p and q.

Finding the order / period

A fast order-finding algorithm allows us to factor numbers quickly.
It remains to figure out how to quickly find the order.

Equivalently, we can look for the period of the sequence

$$a^0, a^1, a^2, a^3, \ldots \pmod{N}$$

Fourier transform is a great tool for finding periodic patterns in data.

Classically, we could use the fast Fourier transform, but this would require time $N \log N$, which is exponential in $\log N$, the number of bits of N.

Discrete Fourier transform

The discrete Fourier transform (DFT) of a sequence of M complex numbers

$$x_0, x_1, \ldots, x_{M-1}$$

is another sequence of M complex numbers

$$y_0, y_1, \ldots, y_{M-1}$$

such that

$$y_j = \frac{1}{\sqrt{M}} \sum_{k=0}^{M-1} \omega^{jk} x_k$$

where $\omega = e^{2\pi i/M}$ is the M-th root of 1.

DFT as a unitary matrix

The discrete Fourier transform is a linear operation on $\mathbb{C}^M$:

$$\begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{M-1} \end{pmatrix} = D \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{M-1} \end{pmatrix}$$

where $D_{jk} = \omega^{jk}/\sqrt{M}$. More explicitly:

$$D = \frac{1}{\sqrt{M}} \begin{pmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \omega^3 & \cdots & \omega^{M-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \cdots & \omega^{2(M-1)} \\ 1 & \omega^3 & \omega^6 & \omega^9 & \cdots & \omega^{3(M-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{M-1} & \omega^{2(M-1)} & \omega^{3(M-1)} & \cdots & \omega^{(M-1)(M-1)} \end{pmatrix}$$

Clearly, $D^\mathsf{T} = D$, so $D^\dagger = \overline{D^\mathsf{T}} = \bar{D}$. One can check that D is unitary by noting that $\bar{\omega} = \overline{e^{2\pi i/M}} = e^{-2\pi i/M} = \omega^{-1}$.

Quantum Fourier transform

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

Peter Shor showed how to implement D using $O\big((\log M)^2\big)$ one- and two-qubit gates. This is polynomial in $\log M$ = the number of qubits!

The $M \times M$ unitary matrix D is therefore also known as the quantum Fourier transform (QFT).

Note: QFT does not give a fast way to *compute* the DFT on a quantum computer, in the sense of obtaining the numbers $y_0, y_1, \ldots, y_{M-1}$. Just like we can't extract all decimal digits of the numbers x_i by measuring a single copy of $|x\rangle = \sum_i x_i |i\rangle$, we can't extract y_i from $|y\rangle = D|x\rangle$ even though we can easily apply D on a quantum computer.

Fourier transform on binary strings

Suppose $M = 2^n$ and let $|x\rangle \in \mathbb{C}^M$ be a computational basis state where $x \in \{0, \ldots, 2^n - 1\}$.

We can uniquely write $x = b_1 2^{n-1} + b_2 2^{n-2} + \cdots + b_n$ for some $b_j \in \{0, 1\}$ (i.e., $b_1 b_2 \ldots b_n$ is the binary representation of x).

One can check that

$$D|b_1 b_2 \ldots b_n\rangle = \frac{1}{\sqrt{2^n}} \big(|0\rangle + \beta_n |1\rangle\big) \otimes \big(|0\rangle + \beta_{n-1} |1\rangle\big) \otimes \cdots \otimes \big(|0\rangle + \beta_1 |1\rangle\big)$$

where

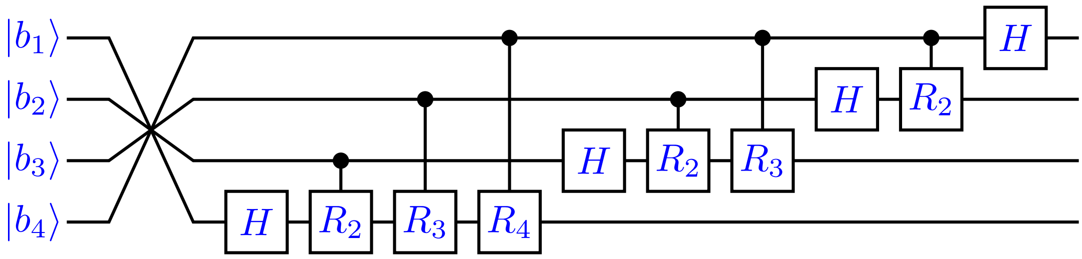
$$\beta_j = \exp(2\pi i 0.b_j b_{j+1} \ldots b_n)$$

and $0.b_j b_{j+1} \ldots b_n \in [0, 1]$ is the binary representation of

$$\frac{b_j}{2} + \frac{b_{j+1}}{4} + \cdots + \frac{b_n}{2^{n-j+1}}$$

Quantum Fourier transform circuit

We can use this form to implement the quantum Fourier transform using Hadamard gates H and conditional phase-shift gates R_k:



$$H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \qquad R_k = \begin{pmatrix} 1 & 0 \\ 0 & e^{2\pi i/2^k} \end{pmatrix}$$

Conditional phase shifts

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.

Period finding

Recall: Given $a, N \in \mathbb{N}$ such that $a < N$ and $\gcd(a, N) = 1$, we would like to find the order of a modulo N, i.e., the smallest integer $r \geq 1$ such that $a^r \equiv 1 \pmod{N}$.

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

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

Note that f_a is periodic, with period at most N. Also note that $f_a(0) = f_a(r)$ is equivalent to $a^r \equiv 1 \pmod{N}$, so the period of f_a is equal to the order of a. How can we find the period of f_a?

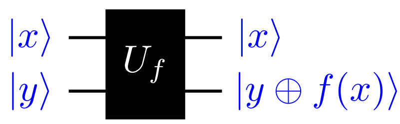
More generally, suppose we can evaluate some arbitrary function $f : \mathbb{N} \to \{0, \ldots, N-1\}$ which is promised to be periodic, i.e., for some integer $r \geq 1$ and all x,

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

How can we find the least value of such r, i.e., the period of f?

Evaluating f in superposition

Let $f : \{0,1\}^n \to \{0,1\}^n$ and U_f be an oracle that reversibly implements f (note that here $x, y \in \{0,1\}^n$ are n-bit strings and so is $f(x)$):



Let us denote the uniform superposition by

$$|\Psi\rangle = H^{\otimes n}|0^n\rangle = |+\rangle^{\otimes n} = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle$$

We can evaluate all values of f in superposition as follows:

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

Note: This does not mean that we can simultaneously extract all values of $f(x)$ from this state. By measuring in the standard basis, we can get each pair $(x, f(x))$ only with exponentially small probability.

The 1st measurement

Measure the second register (i.e., the last n qubits) of $U_f|\Psi\rangle|0^n\rangle$ and denote the outcome by $f_0 \in \{0,1\}^n$. The state after the measurement is:

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

where

- $x_0 \in \{0,\ldots,N-1\}$ is the least value such that $f(x_0)=f_0$
- $r \in \{1,\ldots,N-1\}$ is the period of the function f
- $m = \lfloor 2^n/r \rfloor$ is the number of x such that $f(x)=f_0$

Note: The state in the first register has a periodic structure. We want to extract the period using QFT.

QFT application

We now apply the n-qubit quantum Fourier transform

$$D = \frac{1}{\sqrt{2^n}}\sum_{x,y=0}^{2^n-1}\omega^{xy}|y\rangle\langle x|$$

to the first register (i.e., the first n qubits of the left-over state):

$$\begin{aligned} D\left(\frac{1}{\sqrt{m}}\sum_{k=0}^{m-1}|x_0+kr\rangle\right) &= \frac{1}{\sqrt{2^n}}\sum_{y=0}^{2^n-1}\frac{1}{\sqrt{m}}\sum_{k=0}^{m-1}\omega^{(x_0+kr)y}|y\rangle \\ &= \sum_{y=0}^{2^n-1}\omega^{x_0y}\frac{1}{\sqrt{2^n}}\frac{1}{\sqrt{m}}\left(\sum_{k=0}^{m-1}\omega^{kry}\right)|y\rangle \end{aligned}$$

where $\omega = e^{2\pi i/2^n}$ is the 2^n-th root of 1.

The 2nd measurement

We measure the resulting state in the standard basis:

$$\sum_{y=0}^{2^n-1} \omega^{x_0 y} \frac{1}{\sqrt{2^n}} \frac{1}{\sqrt{m}} \left(\sum_{k=0}^{m-1} \omega^{kry} \right) |y\rangle$$

The probability of observing outcome $y \in \{0,1\}^n \cong \{0,\ldots,N-1\}$ is:

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

This probability distribution peaks at those y for which $ry/2^n$ is close to an integer. Indeed, assuming $ry/2^n$ is exactly an integer (so $rm = 2^n$),

$$\omega^{kry} = \exp(2\pi i kry/2^n) = \exp(2\pi i k) = 1$$

and $p(y) = \frac{|m|^2}{2^n m} = \frac{m}{2^n} = \frac{1}{r}$. In this case, the number of multiples of $r/2^n$ that are integers is r, so we always obtain y that is a multiple of $r/2^n$.

Fact: Given an integer multiple of $r/2^n$, one can recover r using continued fraction expansion.

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 $|f(x)\rangle$.

The probability of successfully finding the period in any run of the algorithm is only ≈ 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.

Summary

- **Factoring:** classically: $O(\exp(\sqrt[3]{\log N}))$, quantumly: $O((\log N)^2)$, where $\log N$ is the input size and N is the number of to be factored
- **Order:** smallest $r \geq 1$ such that $a^r \equiv 1 \pmod{N}$
- **Period:** smallest $r \geq 1$ such that $f(x+r) = f(x)$ for all x; it is equal to the order of a if $f(x) = a^x \bmod N$
- **Reduction:** ability to find orders can be used to factor;
- **Idea:** $x^2 = a^r \equiv 1 \pmod{N}$ so $(x-1)(x+1) = kpq \equiv 0 \pmod{N}$
- **DFT:** $D_{jk} = \omega^{jk}/\sqrt{M}$ where $\omega = \exp(2\pi i/M)$; D is unitary
- **QFT:** its circuit implementation uses the fact that $D|b_1 b_2 \ldots b_n\rangle = \frac{1}{\sqrt{2^n}}\big(|0\rangle + \beta_n|1\rangle\big) \otimes \big(|0\rangle + \beta_{n-1}|1\rangle\big) \otimes \cdots \otimes \big(|0\rangle + \beta_1|1\rangle\big)$ where $\beta_j = \exp(2\pi i 0.b_j b_{j+1} \ldots b_n)$
- **Shor's algorithm:** $(D \otimes I)U_{f_a}|+\rangle^{\otimes n}|0\rangle^{\otimes n}$, measuring the 1st register gives a number that is close to an integer multiple of $r/2^n$; one can find the order r of a modulo N from here; the factors of N are obtained form r and a using the classical reduction