AQI: Advanced Quantum Information
Lecture 2 (Module 4): Order finding and factoring algorithms

February 20, 2013

Lecturer: Dr. Mark Tame
(email: m.tame@imperial.ac.uk)

Introduction

In the last lecture we looked at the quantum Fourier transform, which then led to the quantum phase estimation algorithm. The good news is that we're more than halfway to Shor's algorithm! The bad news is that this bit gets rather technical... anyways, let's see how it goes. I'll start by looking at the order finding algorithm which is a crucial subroutine of Shor's factoring algorithm. The order finding algorithm is sometimes also referred to as Shor's algorithm because it can be used for other things apart from just factoring.

1 Order finding algorithm (Shor's algorithm)

What's an 'order'?

"For positive integers x and N, with $x < N$ and no common factors, *i.e.* $gcd(x, N) = 1$, the order of x modulo N is defined as the least positive integer r such that $x^r \mathrm{mod} N = 1$."

There are two mathematical concepts here that you've probably come across before, but just so we're on the same page, I'll give an example of each:

- The first is the modulo operation, mod. For $20 \bmod 15 = 5$, we chop up the number 20 into chunks of 15 and take the remainder, which gives us 5. Other variants you might see in the literature are $20 \pmod{15} = 5$ and $20 \equiv 5 \bmod 15$. I won't be using any of these!
- The second is the greatest common divisor operation, gcd. For $gcd(5, 21)$ we have that $x = 5$ has the divisors/factors 1 and 5, and $N = 21$ has the divisors 1, 3, 7 and 21. The greatest integer that is a common divisor of both is 1, therefore $gcd(5, 21) = 1$.

Going back to the definition of the order, we have for $x = 5$ and $N = 21$ the order $r = 6$. This can be checked as $5^6 = 15625$ and $21 \times 744 = 15624$, therefore $5^6 \bmod 21 = 1$.

Order finding is believed to be a 'hard' problem on a classical computer: No algorithm is known to solve the problem efficiently. By this I mean using resources polynomial in the $\mathcal{O}(L)$ bits needed to specify the problem. Here, $L = \lceil \log_2 N \rceil$ is the number of bits needed to specify N. However, we'll see in a moment how Shor found a way to do order finding efficiently using a quantum computer.

Order finding is just the quantum phase estimation algorithm applied to the unitary operator U which acts as follows on the computational basis states:

$$U\left|y\right\rangle = \left|xy \bmod N\right\rangle \qquad y \in \{0,1\}^L. \tag{1}$$

Note that for $N \leq y \leq 2^L - 1$ we set $U\left|y\right\rangle = \left|y\right\rangle$ to avoid complications to the outcomes for $y \geq N$. In this case as we know x and N (they're given to us) we know the form of U and can hopefully construct it efficiently somehow (see later). In order to see more closely the connection of order finding to phase estimation we look at the eigenstates of U given by $\left|u_s\right\rangle$

$$\left|u_s\right\rangle = \frac{1}{\sqrt{r}}\sum_{k=0}^{r-1} e^{i\frac{2\pi sk}{r}}\left|x^k \bmod N\right\rangle, \qquad 0 \leq s \leq r-1. \tag{2}$$

For which one finds $U\left|u_s\right\rangle = e^{i\frac{2\pi s}{r}}\left|u_s\right\rangle$ (see Appendix A). Thus we can use phase estimation to extract out $\tilde{\varphi} \simeq \frac{s}{r}$. But once we find $\tilde{\varphi}$ how do we get r?

By choosing the first register (the one that ends up with $|\tilde{\varphi}\rangle$) as having $t = 2L + 1 + \lceil \log_2(2 + \frac{1}{2\epsilon})\rceil$ qubits we get $\tilde{\varphi}$ as an approximation of s/r accurate to $2L+1$ bits (see the end of the phase estimation lecture). In this case $|\frac{s}{r} - \tilde{\varphi}| \leq \frac{1}{2^{2L+1}}$ and since $r \leq N \leq 2^L$ we have $\frac{1}{2^{2L+1}} \leq \frac{1}{2r^2}$ so that

$$|\frac{s}{r} - \tilde{\varphi}| \leq \frac{1}{2r^2}. \tag{3}$$

From number theory we know that if this inequality holds for two rational numbers s/r and $\tilde{\varphi}$, then s/r will be a convergent of the continued fraction for $\tilde{\varphi}$. What's a continued fraction? And what's a convergent?

If a rational number x has a continued fraction representation

$$x = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{\cdots + \cfrac{1}{a_n}}}}, \tag{4}$$

where n is a finite integer and the a_i are integers. The 'convergents' are the rationals $a_0, a_0 + \frac{1}{a_1}, a_0 + \frac{1}{a_1 + \frac{1}{a_2}}, \ldots$ of which there are a total of n. If $x < 1$, then we set $a_0 = 0$. If we have the number $x = p/q$, where p and q are L bit integers then we can find all the convergents using $\mathcal{O}(L^3)$ operations classically (using the continued fractions algorithm).

Once we have all the convergents we have our candidate s' and r' values, where $\frac{s'}{r'}$ are the convergents of $\tilde{\varphi}$ and we can test if $x^{r'} \bmod N = 1$.

Summary (halfway!)

A short recap before we go forward: So far we have that order finding can be achieved by using the phase estimation algorithm which requires $\mathcal{O}(L^3)$ operations to get the order r. There are two main sources of overhead in this:

1. We must be able to efficiently implement controlled-U^{2j} operations for phase estimation.
2. We must be able to efficiently prepare an eigenstate $\left|u_s\right\rangle$ with nontrivial eigenvalue.

Note that in the phase estimation algorithm from in the last lecture we weren't allowed to know U or $\left|u_s\right\rangle$. Now we can know U, but not $\left|u_s\right\rangle$ otherwise we'd know r already!

Modular exponentiation (overhead 1)

The phase estimation algorithm does the following $|j\rangle|u\rangle \overset{QPE}{\rightarrow} |j\rangle U^j|u\rangle$ regardless of $|u\rangle$ being an eigenstate. So we want

$$\begin{aligned} |z\rangle|y\rangle \rightarrow |z\rangle U^z|y\rangle &= |z\rangle U^{z_t 2^0} U^{z_{t-1}2^1} \dots U^{z_1 2^{t-1}}|y\rangle \text{ via step 1 of phase estimation circuit} \\ &= |z\rangle \left| x^{z_t 2^0} x^{z_{t-1}2^1} \dots x^{z_1 2^{t-1}} y \bmod N \right\rangle \\ &= |z\rangle |x^z y \bmod N\rangle . \end{aligned} \tag{5}$$

I've written out the transformation explicitly so we can see the logic operation U^z acting on the computational basis states and work out how we might construct it. From Eq. (5) the sequence of operations used in the quantum phase estimation (step 1) is equivalent to multiplying the contents of the second register by the modular exponential $x^z \bmod N$, where z is the content of the first register. This can be done efficiently using $\mathcal{O}(L^3)$ gates as follows: Compute $x^2 \bmod N$ classically, then square the result and 'mod' it to get $x^4 \bmod N$, then carry on up to $x^{2^{t-1}} \bmod N$. These values are then used to carry out controlled multiplication on the second register $|y\rangle$ dependent on the value of z_{t-1}, z_{t-2}, $\dots z_1$ respectively, as shown in the figure below for z_{t-1}:

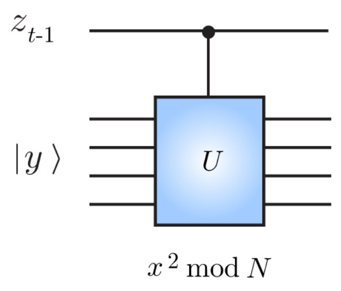

Figure 1: Basic modular exponentiation unit.

Then, we get the desired operation by noting that

$$x^z y \bmod N = ((x^{z_t 2^0} \bmod N)(x^{z_{t-1}2^1} \bmod N)\dots(x^{z_1 2^{t-1}} \bmod N)y) \bmod N \tag{6}$$

Eigenstate preparation (overhead 2)

Note that preparing $|u_s\rangle$ means we must know r beforehand! This isn't possible... but we can use the fact that

$$|1\rangle = \frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} |u_s\rangle . \tag{7}$$

The proof is given in Appendix B.

So we can use $t = 2L + 1 + \lceil \log_2(2 + \frac{1}{2\epsilon}) \rceil$ qubits in the first register and prepare the second register in $|1\rangle$. Then for each s in the range 0 to $r-1$, we get an estimate $\tilde{\varphi} = s/r$ accurate to $2L+1$ bits with probability at least $(1-\epsilon)/r$. Note that in practice we don't need to measure the second register as the first will always have one of the $|\tilde{\varphi}\rangle$ states in it

$$\frac{1}{\sqrt{r}}\sum_{s=0}^{r-1}|\tilde{\varphi}_s\rangle_1\,|u_s\rangle_2\,. \tag{8}$$

Thus, we can 'trace out' (discard) the second register.

Performance

There are some performance issues with Shor's order finding algorithm. Order finding can fail if:

1. Phase estimation gives a bad estimate to s/r. This occurs with probability at most ϵ which can be made arbitrarily small with negligible increase in the size of the circuit.

2. As there is no control over which s we get, s and r may actually have a common factor, so that the r' returned by continued fractions is a factor of r and not r itself! For a randomly chosen $s \leq r-1$ it's very likely that s and r are coprime, *i.e.* $gcd(s,r)=1$, and this doesn't happen. This is because the number of primes less than r is at least $\frac{r}{2\log_2 r}$ due to the Euler function. So the chance that s is prime (and therefore coprime) to r is at least $\frac{r}{2\log_2 r}/r = \frac{1}{2\log_2 r} > \frac{1}{2\log_2 N}$ so if we repeat the algorithm at least $2\log_2 N \leq L$ times we'll get with high probability a phase s'/r' such that $gcd(s,r)=1$ and continued fractions will produce an r as desired. There are are more sophisticated methods to improve the efficieny of this based on feedback iterations.

Summary

The figure below summarises the order finding algorithm.

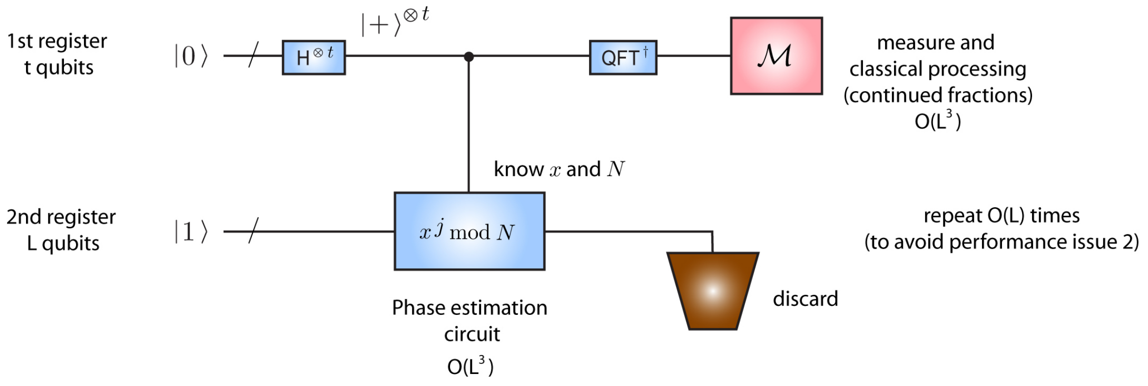

Figure 2: Summary of order finding.

We now have an algorithm for order finding that has a runtime polynomial in the L bits (or qubits for the quantum part) needed to specify the problem, *i.e.* $\mathcal{O}(L^3)$, and a space of $\mathcal{O}(L)$ bits (qubits). The best classical algorithm that can do this has a runtime of $\mathcal{O}(e^{\frac{64}{9}L^{1/3}\log_2 L^{2/3}})$, which grows faster than any power of L, *i.e.* it grows exponentially. Thus we have an exponential speedup! But what can we do with it?...

2 Factoring algorithm (also known as Shor's algorithm)

Shor's factoring algorithm is based on the order finding algorithm and allows one to efficiently factor numbers. This is quite a remarkable result given the context of the following problem:

"Given the positive composite integer N, find two prime numbers p and q which when multiplied together equal it: $N = pq$."

This problem is believed to be intractable on a classical computer, with the best known algorithm being the general number field sieve algorithm, for which the runtime is given by $\mathcal{O}(e^{\frac{64}{9}L^{1/3}\log_2 L^{2/3}})$. Indeed, the security of RSA-based public key cryptosystems, the most widely used encryption method on the internet and in the military, rely on the 'hardness' of this problem. It turns out that this factoring problem is equivalent to the order finding problem, for which we know we can solve efficiently using a quantum computer!

In fact, the equivalence between order finding and factorising is more general than just factoring $N = pq$. Order finding allows us to efficiently find all the factors for a given number N. However, for the present discussion I'll limit this to the specific case of $N = pq$.

The algorithm

1. Pick a random $x \leq N-1$ that is coprime to N. The probability for this to occur is $\geq \frac{1}{2\log_2 N}$ (see the order finding section: performance issue 2). So we only have to repeat $\mathcal{O}(\log_2 N) = \mathcal{O}(L)$ times.

2. Use Shor's order finding algorithm to get the order r for which $x^r \bmod N = 1$. This has a runtime of $\mathcal{O}(L^3)$.

3. Now suppose that r is even and $x^{r/2} \bmod N \neq -1$. The probability for this to occur is $\geq \frac{1}{2}$ if N is odd (which is true as $N = pq$). Let $y = x^{r/2}$, which leads to $(y^2-1) \bmod N = 0$ as $x^r \bmod N = 1$. This gives
$$(y+1)(y-1) \bmod N = 0 \tag{9}$$
Now, $(y-1) \bmod N = (x^{r/2}-1) \bmod N \neq 0$ as $x^k \bmod N = 1$, where $k = r$ is the smallest k for this to hold. This gives
$$(y-1) \bmod N \neq 0. \tag{10}$$
From the relation $x^{r/2} \bmod N \neq -1$ we also have
$$(y+1) \bmod N \neq 0 \tag{11}$$
Due to Eqs. (10) and (11) we know that $(y-1)$ and $(y+1)$ are not divisible by $N = pq$, but Eq. (9) tells us that their product is. As p and q are prime this can only happen if one of them, say p, divides $(y-1)$ and the other, say q, divides $(y+1)$. Take a look at Appendix C to see this. Now the only divisors of N are p and q, and as they are prime, we have that $gcd(y-1, N) = p$ and $gcd(y+1, N) = q$.

So in summary, for a given $N = pq$, once the order r is found for a randomly selected x, we get y and we can find p and q by calculating the gcd of $y-1$ and $y+1$ with N, which can be done using Euclid's algorithm with a runtime of $\mathcal{O}(L^3)$.

Example

Factoring 15:

1. Choose a random number $x < N$ that has no common factors with N. Choose $x = 7$.

2. Compute the order r of 7 mod 15 using the following circuit

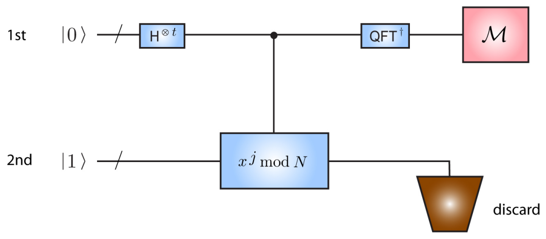

Figure 3: Circuit for order finding subroutine of factoring algorithm

- Consider the following input state $\frac{1}{\sqrt{2^t}}\sum_{k=0}^{2^t-1}|k\rangle|1\rangle = \frac{1}{\sqrt{2^t}}[|0\rangle + |1\rangle + \cdots + |2^t - 1\rangle]|1\rangle$. Use $t = 11$ so that the error probability is at most $1/4$.
- Compute $f(k) = x^k \bmod N$ putting the result in the second register to give

$$\frac{1}{\sqrt{2^t}}\sum_{k=0}^{2^t-1}|k\rangle\left|x^k \bmod N\right\rangle = \frac{1}{\sqrt{2^t}}[|0\rangle|1\rangle+|1\rangle|7\rangle+|2\rangle|4\rangle+|3\rangle|3\rangle+|4\rangle|1\rangle+|5\rangle|7\rangle+|6\rangle|4\rangle+\ldots]. \tag{12}$$

- Assume the second register is measured, obtaining $|1\rangle$, $|7\rangle$, $|4\rangle$ or $|13\rangle$. Note this is not necessary in general and just done here to show the statistics of the outcomes. Say $|4\rangle$ is obtained (any result works), then the input state to the QFT† is

$$\sqrt{\frac{4}{2^t}}[|2\rangle + |6\rangle + |10\rangle + |14\rangle + \ldots]. \tag{13}$$

Applying QFT† we get $\sum_\ell \alpha_\ell |\ell\rangle$ with probability distribution as shown below

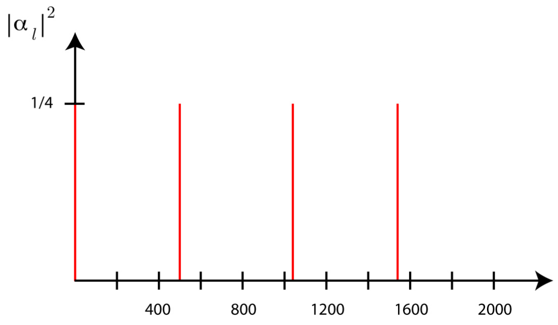

Figure 4: Probability distribution for the output.

This give the states $|0\rangle$, $|512\rangle$, $|1024\rangle$ or $|1536\rangle$ each with probability of $1/4$. Note that $t = 11$ so we can get up to $2^{11} = 2048$ for $|\ell\rangle$.

- Suppose $|1536\rangle$ is obtained, therefore $\tilde{\varphi} = \frac{1536}{2048} = \frac{1}{1+\frac{1}{3}}$, which has the convergents $(0, 1, \frac{3}{4})$. Thus we have our candidate $s' = 3$ and $r' = 4$.

3. By chance r' is even and $x^{r'/2} \bmod N = 7^2 \bmod 15 = 4 \neq -1$, therefore our algorithm succeeded! So we compute $gcd(x^2 - 1, 15) = 3$ and $gcd(x^2 + 1, 15) = 5$. This gives the two primes factors: $3 \times 5 = 15$.

References

- N. D. Mermin, *Quantum Computer Science: An introduction*, Cambridge University Press, Cambridge (2007).
- M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*, Cambridge University Press, Cambridge (2000).
- P. W. Shor, *Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer*, SIAM Rev. **41**, 303 (1999).
- J. Preskill, *Quantum Information lecture notes*, http://www.theory.caltech.edu/people/preskill/ph229/#lecture (2004).
- The quantum algorithm zoo: http://math.nist.gov/quantum/zoo.
- C. Lomont, *The Hidden Subgroup Problem - Review and Open Problems*, arXiv:quant-ph/0411037 (2004).

Appendix A

Here, I show that $|u_s\rangle$ from Eq. (2) is an eigenstate of U:

$$\begin{aligned} U\,|u_s\rangle &= \frac{1}{\sqrt{r}}\sum_{k=0}^{r-1} e^{-i\frac{2\pi sk}{r}} U \left|x^k \bmod N\right\rangle = \frac{1}{\sqrt{r}}\sum_{k=0}^{r-1} e^{-i\frac{2\pi sk}{r}} \left|(x(x^k \bmod N)) \bmod N\right\rangle \\ &= \frac{1}{\sqrt{r}}\sum_{k=0}^{r-1} e^{-i\frac{2\pi sk}{r}} \left|x^{k+1} \bmod N\right\rangle. \end{aligned} \tag{14}$$

I've used a few tricks to go from the first to the second line in Eq. (14): Let $a \bmod m = b$ and $c \bmod m = d$. This means that $a = b + mj$ and $c = d + mk$. Thus we have that $ac = bd + m(dj + bk + mjk)$ and therefore

$$ac \bmod m = bd \bmod m. \tag{15}$$

This means that $(x(x^k \bmod N)) \bmod N \equiv ((x \bmod N)(x^k \bmod N)) \bmod N$ can be found to be equal to $xx^k \bmod N = x^{k+1} \bmod N$, by letting $a = x$, $c = x^k$ and $m = N$ in Eq. (15).

On the second line of Eq. (14) the $k+1$ state now has the phase of k's state, which differs by $e^{-i\frac{2\pi s}{r}}$ for all k. So if we remove this we get back $|u_s\rangle$. Thus $U\,|u_s\rangle = e^{i\frac{2\pi s}{r}}\,|u_s\rangle$ and we can use phase estimation to extract out s/r.

Appendix B

Here, I show that $|1\rangle = \frac{1}{\sqrt{r}}\sum_{s=0}^{r-1}|u_s\rangle$. Taking the right hand side we have:

$$\frac{1}{\sqrt{r}}\sum_{s=0}^{r-1}\left(\frac{1}{\sqrt{r}}\sum_{k=0}^{r-1} e^{-i\frac{2\pi sk}{r}} \left|x^k \bmod N\right\rangle\right) = \frac{1}{r}\sum_{s=0}^{r-1}\sum_{k=0}^{r-1} e^{-i\frac{2\pi sk}{r}} \left|x^k \bmod N\right\rangle. \tag{16}$$

Now apply $\langle 1|$ on this to give

$$\frac{1}{r}\sum_{k=0}^{r-1}\sum_{s=0}^{r-1} e^{-i\frac{2\pi sk}{r}} \langle 1|x^k \bmod N\rangle = \frac{1}{r}\sum_{k=0}^{r-1} r\delta_{k0}, \tag{17}$$

which gives an overlap of 1, so that the state is $|1\rangle$. The last equality is true because $x^k \bmod N$, with $k = r$, is the smallest nontrivial integer k such that $x^k \bmod N = 1$ and as we sum over integers $< r$ only the $k = 0$ case gives a nonzero result $\square$

Appendix C

Let $(y-1) = u$ and $(y+1) = v$. Then from Eq. (9) we have $((y+1)(y-1)) \bmod N = 0$, which means that $N|uv$. Here, $|$ denotes 'divides by'. This means that

$$uv = kN \tag{18}$$

for some integer k. Now suppose that $gcd(v, N) = 1$, then by Bézout's identity (see below) we have integers m and n such that

$$mv + nN = 1. \tag{19}$$

From Eq. (18) we have $muv = mkN$, which from Eq. (19) gives $u(1 - nN) = mkN$, which gives $u - unN = mkN$, which gives $u = (mk + un)N$. Therefore $N|u$. However, we have that $(y-1) \bmod N \neq 0$, therefore $N \nmid u$ and therefore $gcd(v, N) \neq 1$, which means $gcd(y + 1, N) \neq 1$. We can make a similar argument for $(y - 1)$, which leads to $gcd(y - 1, N) \neq 1$.

Thus, if N doesn't divide $(y + 1)$ we have $gcd(y + 1, N) \neq 1$ and N must have a nontrivial factor (common divisor) with both $(y + 1)$ and $(y - 1)$. As $N = pq$ is the only possible decomposition with p and q as the only divisors, we have that

$$\begin{aligned} p &= gcd(y + 1, N) \\ q &= gcd(y - 1, N) \end{aligned}$$

Bézout's identity

If a and b are nonzero integers with gcd of d, then there exist integers x and y such that

$$ax + by = d. \tag{20}$$

For a proof of this you'll need to check out a book on number theory. Note that x and y can be -ve integers, where one is +ve, the other is -ve.