Quantum Information Processing and Quantum Computation 03/12/2015

Lecture 10: Quantum Period Finding and Shor's Factoring

Instructor: Goutam Paul *Scribe: Kaushik Nath*

1 The Period Finding Problem

Let

$$f : \{0, 1, 2, \ldots, M-1\} \to \{0, 1, 2, \ldots, M-1\}$$

be a periodic function of period r, meaning that

$$f(x) = f(x+r) \quad \forall x \in \{0, 1, 2, \ldots, M-1\}$$

and the values $f(x), f(x+1), f(x+2), \ldots, f(x+r-1)$ are all distinct. Suppose also that $M = 2^m$ is a power of 2 and that $r \leq M/2$.

We will summarize a quantum algorithm that finds r in time polynomial in m and in the size of a classical circuit computing f. The importance of the period-finding problem is that, as we will see next time, the integer factoring problem reduces to it, and so the quantum polynomial time for period-finding yields a quantum polynomial time algorithm for integer factoring.

1.1 The Period Finding Algorithm

The quantum method for period finding essentially follows the steps in the algorithm given below.

1. **Create the quantum state $\frac{1}{\sqrt{M}} \sum_x |x\rangle|f(x)\rangle$**

   Let U_f be a unitary transformation on $l = m+m+O(S)$ bits that maps $|x\rangle|0\ldots0\rangle|0\ldots0\rangle$ to $|x\rangle|f(x)\rangle|0\ldots0\rangle$, where S is the size of a classical circuit that computes f. We construct a circuit over l qubits that first applies Hadamard gates to each of the first m qubits. After these operations, starting from the input $|0^l\rangle$ we get $\frac{1}{\sqrt{M}}|x\rangle|0^{l(m)}\rangle$. Then we apply U_f, which gives us the state $\frac{1}{\sqrt{M}}|x\rangle|f(x)\rangle|0^{l(m)}\rangle$. From this point on, we ignore the last $l - 2m$ wires.

2. **Measure the last m bits of the state $\frac{1}{\sqrt{M}} \sum_x |x\rangle|f(x)\rangle$**

   The outcome of this measurement will be a possible output y of $f()$. Let us call x_0 the smallest input such that $f(x_0) = y$. For such an outcome, the residual state will be

$$\frac{1}{\sqrt{[\frac{M}{r}]}} \sum_{t=0}^{[\frac{M}{r}]-1} |x_0 + tr\rangle|f(x_0)\rangle$$

where $[\frac{M}{r}]$ stands for $\lfloor \frac{M}{r} \rfloor$ or for $\lceil \frac{M}{r} \rceil$ depending on x_0. From this point on we ignore the last n bits of the state because they have been fixed by the measurement.

3. **Apply the Fourier transform to the first m bits**

   The state becomes

$$\frac{1}{\sqrt{M}} \frac{1}{\sqrt{[\frac{M}{r}]}} \sum_s \sum_{t=0}^{[\frac{M}{r}]-1} \omega^{(x_0+tr)\cdot s} |s\rangle$$

4. **Measure the first m bits**

   The measurement will give us an integer s with probability

$$\frac{1}{M} \cdot \frac{1}{[\frac{M}{r}]} |\omega^{x_0 s}|^2 \left| \sum_{t=0}^{[\frac{M}{r}]-1} \omega^{(x_0+tr)\cdot s} \right|^2 = \frac{1}{M} \cdot \frac{1}{[\frac{M}{r}]} \left| \sum_{t=0}^{[\frac{M}{r}]-1} \omega^{trs} \right|^2$$

1.2 Analysis of the Algorithm

We will now discuss how to use the measurement done in step (4) in order to estimate r. The point will be that, with noticeably high probability, s/M will be close to k/r for a random k, and this information will be sufficient to identify r, after executing the algorithm a few times in order to obtain multiple samples. The key to the analysis is to understand the probability distribution of outcomes of the measurement in step (4). The analysis is simpler in the special case in which r divides M, so we begin with this special case.

1.2.1 If M is a Multiple of r

Suppose that $q = M/r$ is an integer, and let us call a value s "good", if s is a multiple of q. Note that there are exactly r good values of s, namely $0, M/r, 2M/r, \ldots, M-r$. If s is good, then, for every t, trs is a multiple of M, and so, $\omega^{trs} = 1$, and the probability that s is sampled is $1/r$, and so the good values of s contain all the probability mass of the distribution.

This means that in step (4) we sample a number s which is uniformly distributed in $\{0, M/r, 2M/r, \ldots, M-r\}$, and the rational number s/M , which we can compute after sampling s, is of the form k/r for a random $k \in \{0, 1, \ldots, r-1\}$. After simplifying the fraction s/M, we get coprime integers a, b such that, $s/M = a/b$; if k and r are coprime, then $r = b$, otherwise b is a divisor of r.

If we execute the algorithm twice, we get two numbers s_1, s_2 such that, $s_i/M = k_i/r$ for random k_1, k_2. If we compute the simplified fractions $a_i/b_i = s_i/M$, then each b_i is either r, or a divisor of r, and, more precisely, we have $b_i = r/gcd(k_i, r)$. Now, if $gcd(k_1, r)$ and $gcd(k_2, r)$ are coprime then, $r = lcm(b_1, b_2)$.

This gives us a quantum algorithm that computes r and whose error probability is the probability that picking two random numbers $k_1, k_2 \in \{0, 1, \ldots, r-1\}$ we have that r, k_1, k_2 all share a common factor. The probability that this happens is at most the probability that k_1, k_2 share a common factor, which is at most

$$\begin{aligned}\sum_{p \text{ prime}} \mathrm{P}(k_1 \text{ is a multiple of } p \wedge k_2 \text{ is a multiple of } p) &\leq \sum_{p \text{ prime}} \frac{1}{p^2} \\ &\leq \sum_{n \geq 2} \frac{1}{n} \\ &= \frac{\pi^2}{6} - 1 \\ &< 0.65\end{aligned}$$

So, we have at least a probability of about $1/3$ of finding the correct r, and this can be boosted to be arbitrarily close to 1 by repeating the algorithm several times.

1.2.2 The General Case

For the general case, when r does not divide M, we define a value of s to be good if sr is approximately a multiple of M. It can be shown that there are approximately r good values of s, each with probability approximately approximately $1/r$, so that the measurement at step (4) will give us with good probability a value of s, such that s/M is close to a multiple of $1/r$, from which we will be able to get a divisor of r, and then, by repeating the algorithm several times, the actual value of r.

2 The Order Finding Problem

For $a \in \mathbb{Z}_N^*$, the order of $a \in \mathbb{Z}_N^*$ (or the order of a modulo N) is the *smallest* positive integer r such that

$$a^r \equiv 1 (\bmod N)$$

The order finding problem is to find the order of an element a, given an integer $N \geq 2$ and an element $a \in \mathbb{Z}_N^*$. Classically this problem is hard. Certainly the obvious approach of computing powers of a modulo N until 1 is obtained can take time exponential in the size of N.

3 Reduction of Factoring to Order Finding

We can factor integers efficiently using quantum computers, and that this is the problem solved by Shor's algorithm. Now let us see that the integer factoring problem can be efficiently solved given an algorithm for order finding. In other words, factoring reduces to order finding. The integer factorization problem is to find a prime factorization for $N = p_1^{k_1} p_2^{k_2} \ldots p_m^{k_m}$ for a positive integer $N \geq 2$.

Let us note a few facts, which we will not prove or discuss in any detail. First, if N is a prime number or a prime power (i.e., p^k for some prime p and integer $k \geq 1$), then there are efficient classical algorithms for solving the integer factorization problem in these cases. So, we can imagine that we first run such an algorithm on the input N; if it succeeds then we are done, otherwise we continue on under the assumption that N has at least two distinct prime factors (i.e., $m \geq 2$).

Next, it is enough to have an algorithm that takes a composite N as input and just finds two integers $u, v \geq 2$ such that $N = uv$. If we have such an algorithm, we can run it recursively (interleaved with the classical algorithm for prime powers) to find a complete prime factorization of N.

Again if our goal is to find integers $u, v \geq 2$ such that $N = uv$ for N an even composite number, then we really dont need to work very hard; we simply report $u = 2$ and $v = N/2$.

Algorithm 1 ReduceFactoringToOrderFind(N)

Require: $N > 2$, and odd composite integer which is not power of a single prime
Ensure: u and v such that $N = u \cdot v$

```
1  while (N is not factored) do
2    a ←$ {2, 3, ..., N − 1}
3    d := gcd (a, N)
4    if (d ≥ 2) then
5      return u = d and v = N/d
6    else
7      r := Order(a, N, Z*_N)
8      if (r%2 = 0) then
9        x = a^{r/2} − 1 ≡ ( mod N)
10       d := gcd (x, N)
11       if (d ≥ 2) then
12         return u = d and v = N/d
13       end if
14     end if
15   end if
16 end while
```

Finally, the only case which needs to be addressed is the one for odd N which is not power of a single prime. We assume that there is an efficient algorithm **Order** to solve the order finding problem. Under the assumption **Algorithm 1** gives a solution for the case. The basic idea of why the algorithm works is as follows. Suppose that the random choice of a is in $\mathbb{Z}_N^*$ (which is very likely), and that the order r of a is even. Then

$$a^r \equiv 1(\text{ mod } N)$$

and so $N|a^r - 1$ which implies that $N|(a^{r/2}+1)(a^{r/2}-1)$. It cannot happen that $N|a^{r/2}-1$, because this would mean that r was not then order of a at all. So, if we are lucky that it is

not the case that $N|a^{r/2}+1$, then we know that the algorithm would work. This is because the factors of N are necessarily split between $a^{r/2}+1$ and $a^{r/2}-1$, so computing the gcd of $a^{r/2}-1$ and N would reveal a nontrivial factor of N.

Each iteration of the loop therefore fails to give an answer if either r is odd or r is even but N divides $a^{r/2}+1$. The probability that neither of these events occur is at least $1/2$.