Quantum Algorithms
Exam 2 Solutions

Patrick Canny

June 18, 2019

Due: 2019-05-15

1 Introduction

The goal of this paper is to give a careful specification of Shor's Algorithm. I will give a brief introduction to the algorithm's procedure (tersely noting its major components and their basic functions), discuss the underlying theorems that build the backbone of the algorithm's classical components, give careful descriptions of the quantum components and how they work and finally provide an additional proof of correctness of the algorithm as a contained system.

Please note that throughout the paper, I will sometimes underline particular expressions, i.e.

$$|\underline{a^k}\rangle$$

This notation denotes that what is being referred to is "the binary representation of a^k", rather than a^k itself.

Without further ado, let's take a moment to examine the algorithm at a high level.

2 The Algorithm

At its core Shor's algorithm seeks to efficiently factor numbers. This makes it an intriguing solution to breaking common cryptography schemes such as RSA encryption, which rely heavily on the computational complexity required to factor a large number. Specifically, given some input y, Shor's algorithm will output $d, d \mid y$ with $Prob \geq 1 - \frac{1}{2^{k-1}}$ where k is the number of prime divisors for y(see the "Classical Components" section for more details on why this is the probability for success).

The terse specification of the algorithm is as follows:

(over)

Algorithm 1: Shor's Factoring Algorithm, tersely specified

Input : $y > 1$

Output: d where $d \mid y$

1. Check if y is even ($y \mod 2 = 0$). If it is, output "$d = 2$".
2. Calculate the kth roots of unity for $k \in \{2, 3, \ldots \lfloor \log(y) \rfloor$. If $y = b^k$ for $b \in \mathbb{Z}$ output "$d = b$" and exit
3. Choose uniform $a \in \{2 \ldots y-1\}$ and compute $b = \gcd(a, y)$. If $b > 1$, output "$b = d$" and exit.
4. Compute $r = \mathbf{per}_y(a)$ using a Quantum circuit. If r is odd, then output "y is prime" and exit.
5. Compute $b = \gcd(a^{r/2} - 1, y)$. Output "$b = d$" if $b > 1$ and exit.
6. Output "y is prime" and exit.

As an overview, parts $1-3$ and 5 of the above algorithm leverage classical components, while step 4 is the quantum section.

It is important to describe the concept of $\mathbf{per}_y(a)$ in this problem. $\mathbf{per}_y(a)$ denotes the "period of a with respect to y". More specifically, say we have some periodic function $f : \{0 \ldots m\} \mapsto \{0 \ldots m\}$. This function being "periodic" means that $\forall x \in \{0 \ldots m-t\}$ there is $f(x) = f(x+t)$, but the values of $f(x), f(x+1), \ldots f(x+(r-1))$ are all distinct.

Take $f(x) = a^x$.

$$\begin{aligned} f(x) = f(z) &\Leftrightarrow a^x = a^z \\ &\Leftrightarrow a^{x-z} = 1 \\ &\Leftrightarrow x - z = k * \mathbf{per}_y(a) \\ &\Leftrightarrow x - z \in \{k * \mathbf{per}_y(a) \mod (y) | k \in \mathbb{Z}\} \end{aligned}$$

which equates period-finding to finding a hidden subgroup of $\mathbb{Z}/y\mathbb{Z}$.

This algorithm has polynomial runtime in the size of the log of the input. Specifically, its complexity is in

$$\mathcal{O}((n^2)(\log^2(N))(\log^3(N)))$$

where n is the number of bits in the input, and $N = 2^n$.

Note that depending on the size of the input, the quantum circuit will need to be adjusted slightly, as the number of input qubits indicates the size of the input number.

3 Classical Components

Shor's algorithm utilizes a variety of classical approaches before even entering a quantum component. The reason behind this is to attempt to find a divisor quickly without even needing the quantum portion of the circuit. In practice, these components may be rarely used, as in classic encryption algorithms large prime numbers are used. Additionally, the classical component serves to reduce the problem of finding a divisor to the problem of finding the period, as described above.

As specified by the algorithm's procedure above, it is important to have a way to quickly compute the greatest common divisor(GCD) of two numbers. It is known that the Euclidian Algorithm's complexity

is roughly $\mathcal{O}(n^2)$, so computing the GCD of a given pair of numbers can be done by a classical computer efficiently.

It is also important to note that Shor's Algorithm's success hinges on a result from number theory:

$$\begin{aligned} a^t \equiv 1 \mod y &\Leftrightarrow a^t - 1 \equiv 0 \mod y \\ &\Leftrightarrow (a^{t/2}+1)(a^{t/2}-1) \equiv 0 \mod y \\ &\Leftrightarrow (a^{t/2}+1)(a^{t/2}-1) = k * y \text{ for some k} \end{aligned}$$

The interesting thing about this result is that it implies that a nontrivial divisor of y can be found if t is known in this equation. This is why it is advantageous to design a quantum circuit which is able to efficiently compute the period of $ax \mod y$. Note that this procedure will fail to find a nontrivial divisor of y if t is odd or if $x^{t/2} \equiv -1 \mod y$.

From here, it is possible to show that the procedure will produce a factor of y with probability $\geq \frac{1}{2^{k-1}}$ where k is the number of prime divisors of y. Shor outlines a proof of this fact in his original paper describing this algorithm, which I will paraphrase here:

Proof. Take $y = \prod_{i=1}^{k} p_i^{\alpha_i}$, that is to say that it is a product of k-many unique primes. Take $a_i = a \mod p_i^{\alpha_i}$

Let r_i be $\mathbf{per}_{p_i^{\alpha_i}}(a_i)$, then r is the least common multiple of all the r_i. If we consider the largest power of 2 dividing each r_i, if all of these powers agree the algorithm will fail:

- if all the powers are 1, then r is odd and $r/2$ does not exist.
- if they are all equal and larger than 1, then $x^{r/2} \equiv -1 \mod y$ since $x^{r/2} \equiv -1 \mod p_i^{a_i}$ for all i.

By the Chinese Remainder Theorem, choosing an $x \mod y$ at random is the same as choosing i where $p_i^{a_i}$ is the ith prime power factor of n. This leads to the fact that, since the multiplicative group $(\mod y^\alpha)$ is cyclic for any odd prime power y^α, each of the previously attained powers of 2 has at most a 50% probability of agreeing with any previous power, so when all k of the powers are considered, the total probability is at most $\frac{1}{2^{k-1}}$, so there is at least a $\frac{1}{2^{k-1}}$ chance that the selected value for x is "good" and yields a factor for y with this probability. □

Finally, it is important to consider the probability of calculating the true value of the period given a number of consecutive runs of this period finding algorithm. Note that the quantum period finding procedure returns some value in the form $e^{2\pi i * k/t}$ where t is the period. From here, it is possible to extract the values for k/t, but we suppose that this is represented as an irreducible fraction $\frac{k'}{t'}$. After running the algorithm n times, we will have $\frac{k'_1}{t'_1}, \frac{k'_2}{t'_2} \ldots \frac{k'_n}{t'_n}$. Consider Lemma 13.2 from the textbook:

Lemma 3.1. *If $n \geq 2$ fractions are obtained, then the probability that their least common denominator is different from t is less than $3 * 2^{-l}$*

Due to this lemma, the more times the quantum period-finding circuit is utilized, the more likely the period is to be correctly determined. Of course, there is always a chance that this procedure could fail, but if there is a specified margin of error the algorithm can be run enough times in order to be within this error bound.

4 Quantum Components

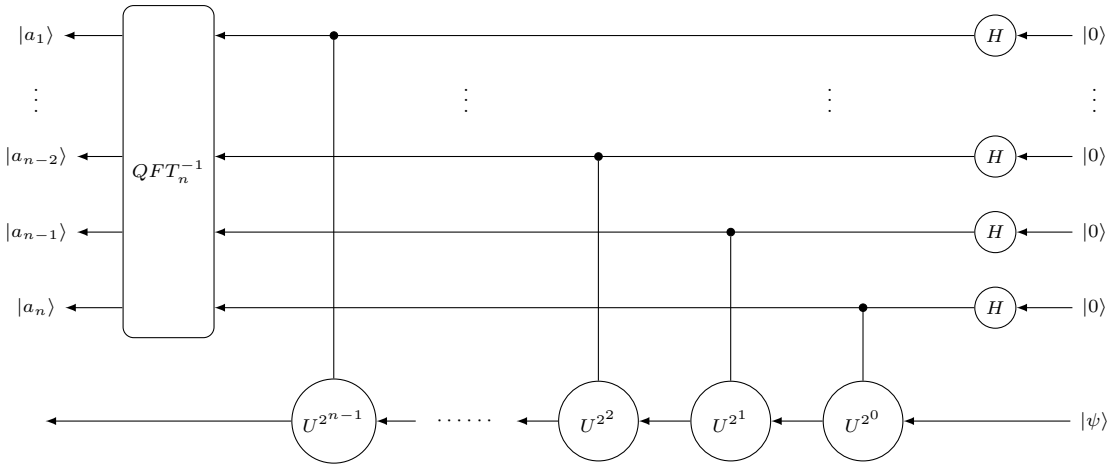
Now that we've shown that if it is possible to efficiently estimate the period of $ax \mod y$, it is possible to effectively factor y, it is time to describe the quantum components of Shor's algorithm.

The quantum components of the algorithm are as follows:

1. A polynomial-size circuit to compute $\mathbf{per}_y(a)$ for $f = ax \mod y$ with low error probability
2. The inverse Quantum Fourier Transform, which is utilized by the period finding algorithm.

So, both of these components will be examined independently, though their results will be consequential in the result of the algorithm as a whole.

1. **Period Finding**
   The quantum circuit for period finding is given as follows:



Where

$$U_a |x\rangle = \begin{cases} |ax \mod y\rangle \text{ if } 0 \leq x < y-1 \\ |x\rangle \text{ otherwise} \end{cases}$$

raised to the various powers associated with it's position in the circuit.

At this point, it is important to talk about the state of this circuit at a variety of points in its execution:

(a) The state directly after the initial application of all the H gates

(b) The state after all consecutive $U_a^{2^i}$ are applied

(c) The state after the application of QFT_n^{-1} is applied, where the top n qubits are measured.

so, each of these states will be considered independently.

(a) The purpose of applying all the H gates to the n input qubits is to create a uniform distribution. We have

$$\begin{aligned} \big(H^{\otimes k} \otimes I\big)\big(\,|0^n\rangle \otimes |\psi\rangle\,\big) &= \big(H\,|0\rangle\,\big)^{\otimes n} \otimes |\psi\rangle \\ &= \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)^{\otimes n} \otimes |\psi\rangle \\ &= 2^{-n/2} \sum_{x \in \{0,1\}^n} |x\rangle \otimes |\psi\rangle \end{aligned}$$

Recall that k was previously selected in the classical portion of the algorithm. This state corresponds directly to the uniform distribution we desire at this point, so now the continued applications of U can be discussed.

(b) First, let's call the application of all the $U_a^{2^i}$, $\Upsilon(U)$. We will fix a $|x\rangle \in \{0,1\}^n$ and look at how the aggregate operator Υ acts on it. Consider the following:

$$\begin{aligned}
\Upsilon\big(\,|x\rangle \otimes |\psi\rangle\,\big) &= |x\rangle \otimes \Upsilon\,|\psi\rangle \\
&= |x\rangle \otimes U_a^{\sum_{i=0}^{n-1} x_i * 2^i}\,|\psi\rangle \\
&= |x\rangle \otimes U^{\underline{x}}\,|\psi\rangle \qquad \text{where } \underline{x} \text{ is the binary representation of the previously selected } x \\
&= |x\rangle \otimes e^{2\pi i(x.\theta)}\,|\psi\rangle
\end{aligned}$$

Where θ has the form $\frac{m}{t}$ and t is the period described above. The reason this last part can be done is because ψ is an eigenvector of U_a, which can be shown by the following calculations.

$$\begin{aligned}
|\xi\rangle &= \frac{1}{\sqrt{t}} \sum_{k=0}^{t-1} e^{-2\pi i k/t}\,\big|\underline{a^k}\big\rangle \\
\implies U_a\,|\xi\rangle &= \frac{1}{\sqrt{t}} \sum_{k=0}^{t-1} e^{-2\pi i k/t} U_a\,\big|\underline{a^k}\big\rangle \\
&= \frac{1}{\sqrt{t}} \sum_{k=0}^{t-1} e^{-2\pi i k/t}\,\big|\underline{a^{k+1}}\big\rangle \\
&= \frac{1}{\sqrt{t}}\big(\,|\underline{a}\rangle + e^{-2\pi i 1/t}\,\big|\underline{a^2}\big\rangle + \ldots + e^{-2\pi i t-1/t}\,|\underline{1}\rangle\,\big) \\
&= \frac{e^{-2\pi i(t-1/t)}}{\sqrt{t}}\big(e^{-2\pi i 1/t}\,|\underline{a}\rangle + \ldots + |\underline{1}\rangle\,\big) \\
&= e^{-2\pi i(t-1)/t}\,|\xi\rangle \\
&= e^{2\pi i(1/t)}\,|\xi\rangle
\end{aligned}$$

where $\big|\underline{a^k}\big\rangle$ denotes that a^k is the binary representation for the given choice of a and k. The step where the power of $\big|a^k\big\rangle$ becomes $k+1$ is because of the definition of U_a: it will multiply a by itself, raising the power by 1. So, we see that U_a has eigenvectors of the form

$$|\xi_k\rangle = \frac{1}{\sqrt{t}} \sum_{m=0}^{t-1} e^{-2\pi i(k*m)/t}$$

with corresponding eigenvalue

$$\lambda_k = e^{2\pi i(k/t)}$$

From here, it is possible to try and deduce the state of the circuit after the application of $\Upsilon(U)$:

$$\Upsilon(U)\,|\psi\rangle = 2^{-n/2} \sum_{x\in\{0,1\}^n} e^{2\pi i(\underline{x}\theta)}\,|x\rangle \otimes |\psi\rangle$$

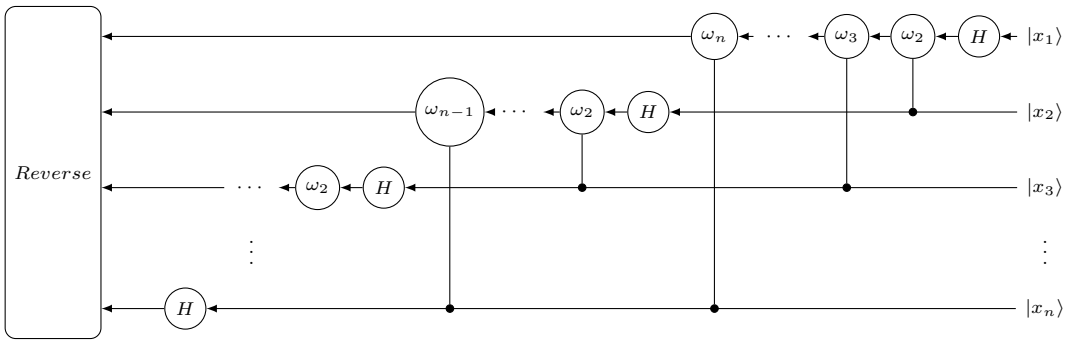
The information we wish to extract from this state is θ as it contains the phase information. This is why QFT_n^{-1} is necessary in this algorithm.

(c) Now that the state directly before entering the QFT block is known, we can now discuss the concrete application of the QFT. We will also discuss what the top n qubits of the circuit represent after this application. The performance of this circuit will be discussed in the next section.

2. **QFT**

Note that in the actual implementation of the phase estimation circuit described above, the inverse Quantum Fourier Transform is utilized. In this section, I will provide a description of a standard QFT, but note that since quantum computation is reversible that it is possible to get QFT_n^{-1} by simply running this circuit backwards.

Now, let's take a look at how the circuit is implemented, and show a few things about it's performance:



Notable here is the fact that if an ω_k gate is shown to be directly on a rail, it really means that the given gate is really being partially controlled by that rail. The way this was denoted in class was to place the gates directly above the rail with two controls, but this has been omitted to save some space.

Also note that this is a very general circuit, and that the patterns introduced here will continue for all of the additional qubits in the sequence.

The QFT_n can be defined in the following way:

$$\mathrm{QFT}\,|x\rangle = 2^{-n/2}\bigotimes_{l=1}^{n}\left(\,|0\rangle + e^{2\pi i(x2^{-l})}\,|1\rangle\,\right)$$

Additionally, $QFT_n^{-1}\,|\gamma\rangle$ can be defined to be

$$2^{-n}\sum_x\sum_y e^{2\pi i(y/2^n)(x-b)}e^{2\pi i(y\delta)}\,|x\rangle\,|\psi\rangle$$

where δ is the error, and $0 \leq b \leq 2^{n-1}, b/2^n = [0\ldots b_n]$

The upshot to utilizing QFT_n^{-1} is that it will be able to extract the values for θ from the previous quantum circuit. This is because of the definition of QFT_n^{-1}:

$$QFT_n^{-1}[1\ldots n]\,|\gamma\rangle = |\theta_1\theta_2\ldots\theta_n\rangle$$

where $|\gamma\rangle$ is the state produced by the phase estimation circuit above, until just before the quantum fourier transform. This is definitely a desirable state to achieve, since it is exactly what will allow the calculation of t with precision noted in the classical components section.

A given application of H will have the following effect on a state:

$$H\,|x_l\rangle = 2^{-1/2}(|0\rangle + \varepsilon_l\,|1\rangle)$$

where

$$\begin{aligned}\varepsilon_l &= \begin{cases} 1 \text{ if } x_l = 0 \\ -1 \text{ if } x_l = 1 \end{cases} \\ &= \begin{cases} e^{2\pi i} \text{ if } x_l = 0 \\ e^{\pi i} \text{ if } x_l = 1 \end{cases} \\ &= \begin{cases} e^{2\pi i[0.0]} \text{ if } x_l = 0 \\ e^{2\pi i[0.1]} \text{ if } x_l = 1 \end{cases} \\ &= e^{2\pi i[0.x_l]}\end{aligned}$$

The addition of the bracketed portions is to illustrate that additional bits of precision are being added with each H. The $[0.1], [0.0]$ are both base 2 numbers, so $[0.1]$ behaves the same as $1/2$. This is what accounts for the addition of the factor of 2 on the first term.

Just after the application of the first ω_2 on the top rail, we can deduce the state and think about how consecutive applications of ω_k will result in a total transformation.

$$\begin{aligned}|\gamma_1\rangle &= \Lambda(\omega_2) H |x_1\rangle \otimes |x_2 \dots x_n\rangle \\ &= 2^{-1/2} \Lambda(\omega_2) \big((|0\rangle + e^{2\pi i[0.x_1]} |1\rangle) \otimes \\ |x_2 \dots x_n\rangle \big) &= 2^{-1/2} \big((|0\rangle + e^{2\pi i[0.x_1 x_2]} |1\rangle) \otimes |x_2 \dots x_n\rangle \big)\end{aligned}$$

Where $\Lambda(\omega_2)$ is the controlled application of ω_2 on the top rail.

From here, the computation can be extended to n applications of ω, resulting in

$$2^{-1/2} \big((|0\rangle + e^{2\pi i[0.x_1 \dots x_n]} |1\rangle) \otimes |x_2 \dots x_n\rangle \big)$$

This result will carry through the other rails, which will then be tensored with the application of every other rail. Specifically, the pattern that emerges after the ω applications on the second rail is:

$$2^{-1/2} \big((|0\rangle + e^{2\pi i[0.x_1 \dots x_n]} |1\rangle) \otimes (|0\rangle + e^{2\pi i[0.x_2 \dots x_n]} |1\rangle) \otimes |x_3 \dots x_n\rangle \big)$$

After all applications, the final state of the circuit is

$$2^{-n/2} \bigotimes_{l=1}^{n} \big(|0\rangle + e^{2\pi i[0.x_{n-(l-1)} \dots x_n]} |1\rangle \big) = QFT |x\rangle$$

This shows that the QFT performs as expected. Because the goal of this circuit in Shor's algorithm is to extract the bracketed portion of the expression, the circuit must be run in reverse to extract $|x_1 \dots x_n\rangle$.

5 Conclusions

Shor's algorithm yields interesting and potentially highly consequential results. The ability to factor numbers in polynomial time could be applied directly to cracking common cryptographic schemes. If quantum computation ever becomes widely used, this would mean that some new form of cryptography would need to be developed to ensure data security. We briefly discussed one such method in class: The k-Local Hamiltonian. The potential issue with this problem is that it is difficult to encode, especially when compared to RSA encryption.

It is important to note that the algorithm as described will find a divisor of a number with high probability. If the total factorization of the number is desired, this algorithm will need to be repeated on the result of dividing the input number by the result of a previous run. Division can be done classically, and is efficient enough to not make an impact on the runtime (I think it's $\mathcal{O}(n^2)$ in the size of the input).