Quantum Computing
Qubits, Factoring, and more

EDWARD HU, CHRIS LONG, AND DAVID MOON
Brown University
May 3, 2016

1 Introduction to Quantum Computing

What is quantum computing? Quantum computing is the study of quantum computers, a theoretical computational system that takes advantage of quantum-mechanical phenomena, most noticeably superposition and entanglement, to perform operations on data. Current computers are mostly electronic and based on transistors. These digital computers take in data through binary bits that is always either a 0 or a 1. However, quantum computers use quantum bits (qubits) which can be in a superposition of states instead of a fixed one.

1.1 Qubits

It's important to understand how qubits relate to quantum computers as they're the fundamental building blocks of quantum computing. Qubits are units of quantum information. Unlike a classical bit, however, a qubit can be in one state, or another state, or a superposition of both! As a result of superposition, each qubit can store more information than a regular bit and multiple bits combined can store vastly more than the same number of bits. However, when measured, the qubit will only display a single state, based on the probabilities of all possible states.

Qubit Representation The two states a qubit can be in are called basis states. Typically, the two basis states are represented as $|0\rangle$ and $|1\rangle$. Because the qubit is a linear superposition of the basis states, it can be represented as a linear combination of $|0\rangle$ and $|1\rangle$

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$$

where α and β are the probability amplitudes of each basis state. α and β are often complex numbers, and the probability of outcome $|0\rangle$ is $|\alpha|^2$ and the probability of outcome $|1\rangle$ is $|\beta|^2$

Entanglement What's key in quantum computing is quantum entanglement. Entanglement is an interesting phenomena that occurs when pairs of particles are generated such that each particle's quantum state are not independent. What that means is that measurements performed on one particle such as spin is correlated with the other entangled particle (regardless of where the other entangled particle is). One particle of the entangled pair seems to somehow become aware of what measurement was made on the other pair, even though it's seemingly impossible for the two particles to have shared any information. Through entanglement, multiple quantum states in a qubit can be acted upon simultaneously unlike classical bits where only one state, 0 or 1, can be acted upon at a time.

2 Shor's Algorithm

Now that we have covered the basics of quantum computing, let's examine a famous quantum computing algorithm. Specifically, we are going to delve into Shor's algorithm, an algorithm which quickly factors numbers and thus has many implications for the future of cryptography and number theory in general.

2.1 What is it?

As previously mentioned, Shor's algorithm allows for swift factorization of composite numbers. That is, it answers the question: given some integer N, can we find a prime factor p such that $p \mid N$? How quick is Shor's algorithm exactly? It has a polynomial runtime (more specifically, according to Wikipedia[1], it has runtime $O((\log N)^2(\log\log N)(\log\log\log N)))$, which means that the amount of time needed to factor N scales like a polynomial as N grows. This is a vast improvement over even the best classical methods of factorization, which are exponential in nature.

2.2 How does it work?

Shor's algorithm has two primary components: reducing the problem to finding period of a function, and the quantum computation of that period. This paper will focus primarily on the first of these steps, as it is the most relevant to the topics we have covered in class. The second step will be discussed in a much more cursory, "big picture" way, since it extends quite a bit beyond the scope of number theory.

The problem Just to reiterate, here is the problem that Shor's algorithm is trying to solve: given an odd composite integer N, find a non-trivial (e.g. not equal to 1 or N) factor p of N.

There are a few restrictions, however. First, N must be odd because all even integers have a factor of 2, making the problem trivial. Second, N must not be a power of a single prime number. This can be relatively easily ensured by checking if any root of N, $\sqrt[k]{N}$, is an integer. We check for all k from 2 to K, where K is the greatest power of 2 less than N; that is $2^K \leq N < 2^{K+1}$.

Both of these cases are easy to factor without needing Shor's algorithm. If all the conditions are satisfied, though, then N is the product of two relatively prime integers (let's call them p and q), and finding those integers becomes much harder.

[1]Wikipedia contributors. "Shor's algorithm." *Wikipedia, The Free Encyclopedia*. Wikipedia, The Free Encyclopedia, 1 May. 2016. Web. `https://en.wikipedia.org/wiki/Shor\%27s_algorithm`.

Reduction to period finding The first step of Shor's algorithm requires reducing the problem of factorization down to the problem of finding the period of a function. Here is an explanation for how this reduction is done.

First, we take the set of all numbers that are relatively prime to N; let's call this set A. The size of A will be $\phi(N)$, where ϕ is Euler's totient function. Next, we take some element $a \in A$ (we can do this at random) and look at the powers of $a \pmod N$:

$$a^1 \bmod N,\ a^2 \bmod N,\ \ldots,\ a^r \bmod N$$

Since $\phi(N)$ is necessarily finite, this sequence must at some point repeat itself. In other words, this sequence must be *periodic* with some period r such that $a^r \equiv 1 \pmod N$ and r is minimal (e.g. there is no $r' < r$ for which this property holds). The goal, then, is to show that if we can successfully find this period, then we have also succeeded in factoring the number itself. If we can accomplish this, then we will have reduced to problem of factoring down to finding the period of a function.

Before we continue, we must note that $1 \pmod N$ has at least 4 distinct square roots. To see this, we need to solve the following two congruences for x

$$\begin{aligned} x^2 &\equiv 1 \pmod p \\ x^2 &\equiv 1 \pmod q \end{aligned}$$

The solutions to this are would also solve

$$\begin{aligned} x &\equiv \pm 1 \pmod p \\ x &\equiv \pm 1 \pmod q \end{aligned}$$

There are 4 different combinations of $+$'s and $-$'s, each of which would give us a unique, nonnegative solution less than pq according to the Chinese remainder theorem. Two of these would be 1 and -1, while the other two would be non-trivial integers between 1 and N. Thus, 1 must have at least two non-trivial square roots mod N (this guarantees the next step will work).

Now let's assume that we have the period r of the function $f(x) = a^x \pmod N$. Furthermore, let's assume that r is even, and that $b = a^{r/2} \not\equiv -1 \pmod N$. If either of these are not satisfied, then take another random $a \in A$ and try again. Furthermore, note that $b \not\equiv 1 \pmod N$, because otherwise r would not be minimal, contradicting our definition of period. Once we have satisfactory a and r, then our claim is that $gcd(b+1,\ N)$ and $gcd(b-1,\ N)$ are both proper factors (e.g. not equal to 1 or N) of N. Again, note that b must exist for some a by the above argument.

Let's prove that they are. If we have valid choices of a and r, then we find that

$$\begin{aligned} b^2 = a^r \equiv 1 \pmod{N} \\ b^2 - 1 \equiv 0 \pmod{N} \\ (b+1)(b-1) \equiv 0 \pmod{N} \end{aligned}$$

which then implies that $N \mid (b+1)(b-1)$. Now let's look at $g = gcd(b+1,\ N)$. Our claim is that $g \neq 1,\ N$. If $g = N$, then $N \mid b+1$, and $b \equiv -1 \pmod{N}$, which contradicts our assumptions on b. Similarly, if $g = 1$, then we choose some $u,\ v$ such that

$$\begin{aligned} u(b+1) + vN = 1 \\ u(b+1)(b-1) + vN(b-1) = b-1 \end{aligned}$$

Then, since $N \mid (b+1)(b-1)$ and $N \mid N$, we can conclude that $N \mid u(b+1)(b-1) + vN(b-1) = b-1$, which then implies that $b \equiv 1 \pmod{N}$, again contradicting our definition of b. Thus, $gcd(b+1, N)$ can be neither 1 nor N, and is therefore a non-trivial factor of N. A very similar argument can be made for $gcd(b-1, N)$.

So, just to recap: if we can find the period of $f(x) = a^x \pmod{N}$, and the period satisfies certain conditions, then we can very easily find two nontrivial factors of N by computing GCDs. Computing GCDs with Euclid's algorithm is extremely fast, so the only real bottleneck here is finding the period of f.

Quantum computation So far, everything we've covered about Shor's algorithm can be computed relatively easily with classical computers. However, finding the period of f can be quite hard for large values of N, since traditional computers essentially have to compound a on itself until it completes a period. This is where quantum computers come in (theoretically, at least).

With quantum computers and their ability to superposition qubits, however, we should be able to quickly and with high probability compute the period of this function. Therein, though, lies another issue with quantum computing; for all the increases in efficiency and flexibility that superpositioning grants us, we can still only observe a single result from the qubits. Thus, there exists the possibility, however miniscule, that the observed result is incorrect. We can statistically render this possibility near impossible, but it is still technically there.

Now, onto how we can use quantum computing to actually find the period of

$$f(x) = a^x \pmod{N}$$

To do this, we first create tuples in the form $(x, f(x))$ where x is an integer in the range $1 \leq x < Q$ and $Q = 2^q$ such that $N^2 < Q < 2N^2$. Using quantum magic, we superimpose

all Q of these tuples together (we create this many to guarantee we have many full periods, so we can find it with very high probability). Having done this, we should have all the following superimposed together:

$$(1,\ a^1 \bmod N),\ (2,\ a^2 \bmod N),\ \ldots\ ,\ (Q-1,\ a^{Q-1} \bmod N)$$

Next, we apply a quantum Fourier transform (to be honest, none of us has any idea what this does), which, as far was we can tell, increases the probabilities of the states which lie on a period point and decreases all other probabilities. Thus, when we go to measure the superposition of qubits, we are vastly more likely to get the period of f, or a multiple of it, rather than some other value. With the period in hand, we can just apply the above process to find factors of N.

Going back to the quantum Fourier transform for a moment. Although none of us fully grasps the technical details of the process, we did find several useful analogies online that illustrated the purpose it played. The first of these likened the process to that of deducing the time schedule an eccentric person followed[2]. Here's the setup: there's an eccentric person who decides that, rather than the usual 24 hour day, he's instead going to follow, for example, a 26 hour day. In his home, however, he keeps clocks of all different day lengths. He may have a usual 24 hour clock, in addition to a 26 hour clock (the one he follows), perhaps as well as 25, 28, and 29.8 hour clocks. Underneath each clock is a posterboard with a thumbtack stuck in the middle. Furthermore, when this person gets up every day (assuming it's at the same time, relative to his 26 hour day), he moves the thumbtack in each posterboard one inch in the direction that the hour hand on the corresponding clock is pointing. The question now is: if you were to step into this room at some arbitrary time, would you be able to tell what day length this person was living on? Of course you would! Since they get up at the same time everyday, the thumbtack corresponding to the time they're following would be much farther from the center than any of the other ones, since it gets moved a inch in the same direction everyday. All other times, however, sort of "swirl" around near the center, since the hour hands point in a different direction each "morning".

This is essentially what the quantum Fourier transform does to the superposition. For every tuple that happens to fall on a periodic point, the probability of observing that point amplifies. For every other point, it has some arbitrary remainder $\pmod N$, and therefore undergoes destructive interference, which decreases the probability of observing it. This is obviously a very, very high level overview of the quantum computation of the period, but none of us really understand the physics or mathematics behind it, and the technical details are not particularly number theoretical in nature.

[2]Aaronson, Scott. "Shor, I'll Do It." *ShtetlOptimized*. WordPress, 24 Feb. 2007. Web. `http://www.scottaaronson.com/blog/?p=208`.

3 Quantum Computing Implications

The rise of quantum computing and its ability to perform signficantly faster on problems relative to our current best classical algorithms offers a lot of future innovation for science and math. For example, quantum computers will be able to better model quantum physical processes from chemistry and solid state physics, as well as the approximation of Jones polynomials. While quantum computing will undoubtedly affect many areas such as quantum simulation, two main areas quantum algorithms are going to affect are cryptography and search.

3.1 Quantum Factoring and Cryptography

With quantum factoring such as Shor's algorithm, many cryptographic systems that rely on the computational difficulty of integer factorization (i.e. RSA) will become much less secure. However, other cryptographic systems such as lattice-based cryptosystems are not affected by quantum algorithms and may become more prevalent as quantum computing becomes more commonplace in the future.

3.2 Quantum Search

Outside of factorization, quantum algorithms have also been promised to have a significant speed up on certain classical algorithms. The best known example is Grover's algorithm, which can used in quantum database search and uses quadratically fewer amounts of tries to find what is desired from the database. Furthermore, Grover's algorithm can obtain a quadratic speed-up over a brute-force search for all NP-complete problems.

3.3 Roadblocks

Unfortunately, there are still a number of challenges that stand in the way from a large-scale fully functional quantum computer. The problem lies in the ability to read qubits easily and conveniently, being able to scale in number of qubits, and being able to remove quantum decoherence (disruption of entanglement). When entangled particles interact with the environment such as being measured, they no longer are entangled which is often necessary in quantum algorithms. However, researchers are currently improving on how we measure and modify qubits to make quantum computing more viable.

References

1. Aaronson, Scott. "Shor, I'll Do It." *ShtetlOptimized*. WordPress, 24 Feb. 2007. Web. `http://www.scottaaronson.com/blog/?p=208`.

2. Wikipedia contributors. "Shor's algorithm." *Wikipedia, The Free Encyclopedia*. Wikipedia, The Free Encyclopedia, 1 May. 2016. Web. `https://en.wikipedia.org/wiki/Shor\%27s_algorithm`.