Shor's Algorithm

Phys 4831 Thomas Alexander
Advanced Quantum Mechanics
Mount Allison University 8 April 2014

1 Prime Factorization

In the modern day world the Internet permeates everything. All of the corners of the world are connected and information is freely exchanged. However, often there is a need to exchange information privately. Unfortunately, the design of the Internet does not make this easy as it is a peer to peer network in which there are no assurance provided that outsiders are not eavesdropping on ones network traffic. To solve this problem, cryptosystems were developed that allow information to be encrypted by anyone and decrypted by only the person holding the keys to the encryption. The RSA algorithm was one of the first and most popular public-key encryption systems. The RSA algorithm generates a public key composed of the product of two very large prime numbers. This public key may be released to anyone and data maybe be encrypted using a public key. However, to decrypt the key requires the two prime numbers used to generate the key. One might wonder if it is possible to calculate what these prime numbers are. The answer to this question is yes! The follow up question to this might be, "than how is this securing the data" and the answer to this is while factoring large prime numbers is completely possible it is very very time consuming! The fastest classical prime number factorization algorithm, the General Number Field Sieve has a complexity of [2]

$$O\left(exp\left(\left(\frac{64}{9}\right)^{\frac{1}{3}}+O(1)\right)(ln(n))^{\frac{1}{3}}(ln(ln(n)))^{\frac{2}{3}}\right) \tag{1}$$

This is a super polynomial algorithm and for very large numbers performs very poorly. For a large number composing of say 2048 bits, it is completely infeasible that the prime factors could be found with today's hardware and even if it were possible an ever larger number could be used and it would be exponentially harder to discover the new keys factors. While there are many other uses for prime numbers, encryption arguable plays the largest role in modern day society. The ability to factor large prime numbers would be game changing. While no polynomial time classical algorithm exists, Peter Shor proposed an algorithm in 1995 that utilizes the physics of quantum mechanics to calculate prime factor of a number.[3] Shor's algorithm can factor a number with

$$O(n^2 log(n) log(log(n))) \tag{2}$$

quantum gates. This is a polynomial time algorithm and has a much smaller complexity than the general number field sieve. The implementation of this algorithm would allow most modern day encryptions to be cracked!

2 Reduction to Period Finding

Shor's algorithm is a hybrid classical and quantum algorithm. The majority of the algorithm is performed on a classical computer, however the large speedup in the algorithm is due to the quantum component. All of the classical components can be performed in polynomial or sub-polynomial time.[2]

The goal of Shor's algorithm is that for an odd composite number N to find an integer d, $1 < d < N$ that divides N. It is also required that N not be the power of a prime number. Consider the choice of some number $1 < x < N \;\; x \in \mathcal{Z}$ if the $gcd(x, N) \neq 1$ than a nontrivial factor has been found, therefore are number N must be composed of two co-prime numbers a, b with a $gcd(a, b) = 1$. Examining the sequence

$$x^0 mod N, x^1 mod N, \cdots x^p mod N$$

Note that $X^0 = 1$. After some number of iterations $0 < r < N$, it can be proven that $x^r mod N = 1$ again. The number of iterations r is known as the period. Therefore we can solve

$$x^r = 1 mod N$$

$$(x^{\frac{r}{2}})^2 = 1 mod N$$

$$(x^{\frac{r}{2}} + 1)(x^{\frac{r}{2}}) - 1) = 0 mod N$$

$$(x^{\frac{r}{2}} + 1)(x^{\frac{r}{2}}) - 1) = kN \;\; k \in \mathcal{Z}$$

Notice that

$$\frac{(x^{\frac{r}{2}} + 1)(x^{\frac{r}{2}}) - 1)}{k} = N$$

As $k > 1, x > 2$ both $(x^{\frac{r}{2}} + 1)$ and $(x^{\frac{r}{2}}) - 1)$ must be greater than zero. As r is the smallest possible integer such that $x^r = 1 mod N$, N cannot possible divide $(x^{\frac{r}{2}} - 1)$ and if it also does not share a common factor with $(x^{\frac{r}{2}} + 1)$ it must share a common factor with both terms. Therefore $gcd((x^{\frac{r}{2}} + 1), N)$ and $(x^{\frac{r}{2}} - 1)$ can be calculated classically to find two non-trivial factors of N.[1]

It has therefore been shown that all that is needed to find a non-trivial factor of N is the period r for some integer $1 < x < N$. For the factorization it is required that r be even, this has been proven to be the case with probability $1/4$ and if r is not even, a new x can always be selected for testing.[2]

3 Period Finding

While we now known how to compute non-trivial factors of N, it is contingent on knowing the period r of $x^r mod N = 1$. Peter Shor created a period finding algorithm that exploited the superposition and entanglement of quantum states, that is well known in quantum mechanics.[1]

Consider our function $f(r) = x^r$, where x is chosen randomly. $f(r)$ must be constructed as a quantum function to give a superposition of all the results. We create a quantum superposition of

states using two quantum registers for all $1 < r < N$.[2]

$$|\Psi_1\rangle = \frac{1}{\sqrt{2^N}} \sum_{r=0}^{N-1} |r\rangle\,|f(r)\rangle$$

The classical discrete Fourier Transform takes a vector of complex components $\{f(0), f(1), \cdots, f(2)\}$ to a new complex vector $\{\tilde{f}(0), \tilde{f}(1), \cdots, \tilde{f}(2)\}$ defined by

$$\tilde{f}(k) = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} e^{2\pi i \frac{jk}{N}} f(j)$$

The quantum Fourier transform does this same thing to a quantum register of n qubits , $N = 2^n$ and in operator form is [2]

$$F(|j\rangle) = \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} e^{2\pi i \frac{jk}{2^n}} |k\rangle$$

The Quantum Fourier transform can be constructed from $O(n)$ swap gates and is therefore efficiently realizable with a universal quantum computer.[2] Applying the Quantum Fourier transform operator to the input register $|r\rangle$ of $|\Psi_1\rangle$ results in

$$|\Psi_2\rangle = F\,|\Psi_1\rangle = \frac{1}{N} \sum_{r=0}^{N-1} \sum_{k=0}^{N-1} e^{\frac{2\pi i r k}{N}} |k\rangle\,|f(r)\rangle$$

Now measurement is performed both on the input register $|r\rangle$ and the outcome register $|f(r)\rangle$ with outcomes k and $f(r_o)$ respectively. As the function $f(r)$ is periodic, the probability to measure any given k for $f(r_0)$ is given by[1]

$$\langle f(r)\,|\,\Psi_2\rangle = \frac{1}{N^2} \left| \sum_{r:f(r)=f(r_o)} e^{\frac{2\pi i r k}{N}} \right|^2 = \frac{1}{N^2} \left| \sum_b e^{\frac{2\pi i (r_0 + r_p b) k}{N}} \right|^2$$

Where r_p is the period. If $\frac{r_p k}{n}$ is an integer the chance of detection is 100%, and decreasing the farther it is away from being an integer. Therefore if a measurement is performed for k and it is detected, the relationship $kr_p/N = m \;\;,\; m \in \mathcal{Z}$ should be very close to being accurate. From this it follows that

$$\frac{k}{N} = \frac{m}{r_p}$$

By converting the left hand side to an irreducible fraction, the denominator of this fraction will be the period r_p with large probability. Irreducibility will Verification that $f(r) = f(r + r_p)$ must be done as it is not guaranteed that the discovered r_p is the period due to the issue of measurement in quantum mechanics. If r_p is not the true period, other selections of y near the original y and multiples of r_p may be placed in irreducible form and tested as they have are also realistic candidates. If no realistic candidates are found, the algorithm got unlucky and must be restarted by restarting the period finding algorithm.

4 Simulating Shor's Algorithm

Shor's algorithm may be simulated on a classical computer, albeit without the polynomial running time of the algorithm implemented on a quantum computer. This is because on a classical computer it is not possible to exploit the exponential state space that exists in the quantum computational model. When entangling qubits the number of states scales as 2^n, where n is the number of qubits being utilized. However, on a classical computer it would require 2^n bits to represent the same state as on the quantum computer and just as many additional operations to manipulate the state space. Therefore, it is expected that the simulated algorithm will run much slower than the best classical algorithms for factorization. However, it is still a useful exercise to implement the algorithm classically.

The general steps in the algorithm for factoring an integer N are as follows and will be implemented in python, in the accompanying Ipython notebook.

1. Choose a number randomly $a < N$
2. Calculate the greatest common denominator of a and N. If it is not the trivial case of 1 than a factor has been found.
3. If no gcd is found than use the Shor's period finding routine to calculate the period of

   $$f(x) = a^x mod N$$

   , ie. when $f(x + r) = f(x)$
4. If r is found to be odd, return to step one.
5. If $a^{\frac{r}{2}} = -1 mod N$ return to step 1.
6. Two factors of N are the $gcd(a^{\frac{r}{2}} \pm 1, N)$
7. If these are not the only two factors of N divide N by the two discovered factors and return to step one with $N_2 = \frac{N_1}{gcd(a^{r/2}+1,N)gcd(a^{r/2}-1,N)}$

References

[1] Shor's factoring algorithm. (2005, September 13). In Quantiki Wiki. Retrieved 21:06, April 6, 2014, from http://www.quantiki.org/wiki/Shor's_factoring_algorithm

[2] Benenti, G., Casati, G., Strini, G. (004). Principles of quantum computation and information. (1 ed.). World Scientific.

[3] Shor, P. W. (1995). Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. ARXIV, doi: arXiv:quant-ph/9508027