A Brief Introduction to Quantum Computing Algorithms

Kevin Anderson

November 23, 2015

1 Introduction

The field of quantum computing is relatively young and has the (perhaps dubious) distinction of being the intersection of physics and theoretical computer science. Unfortunately, this means that much of the available literature is dense and requires extensive knowledge of both fields to grasp. This paper's purpose is to provide an introduction to quantum computing algorithms at a level appropriate for a physics undergraduate student.

First, we'll introduce the computer science background necessary to understand this paper. Algorithms are often contrasted by their "computational complexity" (also called "runtime complexity"), which is a measure of how the number of computations an algorithm requires to solve a problem grows with the size of the problem. For example, the Discrete Fourier Transform (DFT) algorithm takes in an array of N numbers and has a nested loop over the array, requiring at least N^2 computations. In comparing the computational complexity of two algorithms, we only care about the "order" and not arbitrary constants. Perhaps one algorithm requires exactly N^2 operations while another requires $2N^2$ for the same input; these two algorithms both have time complexity denoted by $O(N^2)$. The Fast Fourier Algorithm is famous for having runtime complexity of only $O(N \log N)$, which is a substantial improvement over the DFT. To illustrate the difference, imagine if the input were 2^{32} elements long. The DFT would require something like 10^{19} operations, while the FFT needs only 10^{11} operations – a dramatic decrease.

It is a well-known problem that simulating quantum mechanical systems on a classical computer is difficult. Some systems may have symmetries or other simplifying characteristics that allow efficient simulation on a classical computer, but an arbitrary quantum system requires exponentially more resources to simulate as the system grows in complexity. Suppose we had a quantum system that acted as a computer running some process. If the quantum computer could run an algorithm in, say, $O(N)$ time, then the classical computer would require at least $O(Ne^N)$ time to simulate the quantum computer running the algorithm. This suggests that a quantum computer could offer significant runtime complexity speedup over a classical computer for a given problem, though of course is not proof: simulating a computer solving a problem may not be the best way to solve the problem.

Another tool from computer science is proving bounds for runtime complexities. A given problem can have any number of algorithms that solve it, each with a different runtime complexity. However, based on the structure of the problem, lower bounds on the runtime complexity of any algorithm solving the problem can be established. This is a mathematically rigorous process that won't be covered here, but it is useful to prove that a given algorithm is "optimal" in the sense that no other algorithm solving that problem could have a faster runtime complexity.

2 Qubits

Just as classical computers have a way to physically store information (typically, bits encoded as voltages), quantum computers require some medium of information storage. The crucial difference is that it stores quantum mechanical information (superpositions) rather than classical information (true/false).

As a physical information storage medium, take any quantum-mechanical two-state system. The spin state of an electron is an obvious candidate, but many other such physical systems exist: the $n = 1$ and $n = 0$ states of the quantum harmonic oscillator, the presence/absence of an electron on a quantum dot, and the polarization of a photon are all systems that could be used as qubits. For convenience, the two states are called $|0\rangle$ and $|1\rangle$, so in general the qubit can be in any superposition of $|0\rangle$ and $|1\rangle$. Just as classical computers need many individual bits to be very useful, quantum computing usually focuses on groups of qubits called registers. A register of n bits is in a superposition of $N = 2^n$ different states at the same time, and any operations performed on the register act on all basis states simultaneously. It is this exponential relationship between number of bits and information storage that allows the exponential complexity speedup seen in many quantum algorithms.

When doing math with qubits, it is often convenient to write them as kets or column vectors, as with any quantum mechanical state. We then perform operations on them with matrices representing quantum mechanical operators. Because a quantum computer must have a finite number of bits, we get to use standard finite-dimensional linear algebra, as seen in the next section.

3 Quantum Gates

Any classical algorithm can be represented in circuit/gate form (to prove this, remember that the computers running these algorithms are made from logic gates). This is usually very tedious and the algorithm is better represented in terms of pseudocode. However, the literature has tended towards using circuit diagrams to represent quantum algorithms, and it proves useful in some cases. Here, we'll just use the gate symbols and stick with equation format.

First, lets examine gates that act on a single qubit. Probably the simplest quantum gates are the rough analogs of the classical NOT gate. Written in matrix form, the bitflip gate X and phaseflip gate Z are

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

Note that X swaps the rows of a column vector, causing a pure 1 to flip to 0 and vice versa, while Z negates 1. Another common gate is the phase rotation gate:

$$R_\theta = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{pmatrix}$$

which rotates the phase of 1. Z is the special case of R_θ where $\theta = \pi$.

Note that all of these gates are *unitary*, shown by their conjugates being their inverses: $X^\dagger X = Z^\dagger Z = R_\theta^\dagger R_\theta = I$. Unitary matrices preserve magnitude in the sense that if applied to a vector of total magnitude 1, the result will also be total magnitude 1. In fact, all quantum gates must be unitary, giving us all kinds of useful properties. Unitary gates are also reversible – no information is lost in their application, and so any quantum algorithm can also go in reverse. This has interesting implications for applying information theory and thermodynamics to quantum computers (recall that a thermodynamically reversible process has no increase in entropy), but that is outside the scope of this paper.

In practice, most quantum algorithms rely on more sophisticated gates. One is the Hadamard gate:

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

Note that this paper will never use a Hamiltonian operator, and so there is no ambiguity in the notation H. The Hadamard gate applied to a pure 0 or 1 creates an equal superposition of 0 and 1 with different phases.

Another gate is the Controlled-NOT (CNOT) gate. It takes two qubits as input and negates the second iff the first qubit is 1. In matrix form:

$$U_{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

For some two-qubit state $|\psi\rangle = \alpha\,|00\rangle + \beta\,|01\rangle + \gamma\,|10\rangle + \delta\,|11\rangle = [\alpha, \beta, \gamma, \delta]^T$:

$$U_{CNOT}\,|\psi\rangle = [\alpha, \beta, \delta, \gamma]^T = \alpha\,|00\rangle + \beta\,|01\rangle + \delta\,|10\rangle + \gamma\,|11\rangle$$

Here we see that if the control bit is 0, then $\delta = \gamma = 0$, and the second bit remains unchanged. If the control bit is 1, then $\alpha = \beta = 0$, and the coefficients on the second bit are swapped.

4 Quantum Algorithms

Quantum algorithms typically are assumed to have access to a "black-box" function called an oracle that is specific to a given problem. The oracle's implementation details are ignored and the computational complexity of the quantum algorithm is implicit in terms of the complexity of the oracle, since the algorithm itself is agnostic to the specific details of a given problem. In other words, an algorithm that has linear runtime with a constant-time oracle is classified as linear for all oracles, regardless of the oracles' complexities.

A standard quantum algorithm template is to initialize a register in some state, then repeatedly apply an operator to it that causes destructive interference to the undesired register states while increasing the amplitude of the desired state. After some number of iterations, the probability of measuring the desired state in the register is sufficiently high to be confident in the result. The tricky part of designing such an algorithm lies in the operator: it will be different for every problem, and if it is to be of practical use, it must be implementable on physical hardware.

One sticking point is that many quantum mechanical algorithms are inherently probabilistic, meaning that steps can be taken to minimize the risk of a wrong answer (for instance, more iterations), but the risk can never be completely eliminated. Error correction in quantum algorithms will be discussed later, but for now it is sufficient to say that the runtime complexity is not affected, and so we can proceed.

4.1 Grover's Algorithm

Imagine we have a black-box function that takes in n bits and returns a 1 or 0:

$$f : \{0,1\}^n \to \{0,1\}$$

And there is a special bitstring input $|\phi\rangle$ such that $f(\psi) = 1$ iff $|\psi\rangle = |\phi\rangle$ and $f(\psi) = 0$ otherwise. This problem is equivalent to brute-forcing a password, or searching an unsorted database. The best classical algorithm to solve this problem has linear runtime complexity ($O(N)$, $N = 2^n$ is the size of the database) and is sketched in pseudocode here:

Algorithm 1 Classical Database Search Algorithm

```
Require: $f(i) = 1$ for exactly one $i \in \{0,1\}^n$
1: procedure DB–Search
2:     for each entry $i = 0$ to $N - 1$ do
3:         if $f(i) = 1$ then return $i$
4:         end if
5:     end for
6: end procedure
```

This is common sense: check each possible value until you find the correct one. Because we don't know any additional information about the problem (e.g., the database is sorted), this method will call f an average of $N/2$ times, and so its runtime complexity is $O(N)$. In this case, it is probably obvious that a classical algorithm can do no better than $O(N)$ in solving this problem, but, as discussed above, it can be mathematically shown that no algorithm can solve this problem faster than linear time.

Now, let's see what happens with a quantum algorithm. First, we'll need a few definitions. Following the literature, (Grover, 1996; Shor, 2000), we have $W \equiv H^{\otimes n}$, which applies the Hadamard gate to each qubit in the register. The single-bit Hadamard gate presented above generalizes to

$$H^{\otimes n} |x\rangle = \frac{1}{2^{n/2}} \sum_{y=0}^{N-1} (-1)^{x \cdot y} |y\rangle$$

where $x \cdot y$ is the bitwise dot product of x and y.

We define $|s\rangle$ to be the uniform superposition across all possible states of the register:

$$|s\rangle = \frac{1}{\sqrt{N}} \sum_{i=1}^{N} |\psi_i\rangle$$

Now define $Z_0 = I - 2|0^n\rangle\langle 0^n|$. This operator has the effect of taking $|0^n\rangle$ to $-|0^n\rangle$ and leaving all other basis vectors unchanged. If we form the composite operator $D = WZ_0W$, it can be shown that this is equivalent to $D = 2|s\rangle\langle s| - I$.

Finally, the operator $U = I - 2\left|\phi\right\rangle\left\langle\phi\right|$. At first glance, it might seem like cheating to use $\left|\phi\right\rangle$ in an operator, but consider this: U can equivalently be defined as negating its operand if $f(\psi) = 1$ and leaving it unchanged otherwise. We are given f as an oracle, and we can certainly design an operator that matches this second definition, so we might as well use the first definition if it's more convenient.

As for why we choose these operators, it more or less comes down to the insight of the algorithm designer. For now, let's just see if it works.

The idea behind Grover's Algorithm is, as mentioned above, to start with a uniform superposition in the register and repeatedly cause interference that moves the register's state closer to the final answer. This is done by first initializing the register to $\left|s\right\rangle$ and then iteratively applying U and then D. After some number of iterations, we will have reached our answer, and the register will be measured to be in the state $\left|\phi\right\rangle$ with some near-unity probability. To illustrate this, we examine an iteration of the algorithm. First, apply U to $\left|s\right\rangle$:

$$\begin{aligned}
U\left|s\right\rangle &= \left(I - 2\left|\phi\right\rangle\left\langle\phi\right|\right)\left|s\right\rangle \\
&= \left|s\right\rangle - 2\left|\phi\right\rangle\left\langle\phi|s\right\rangle \\
&= \left|s\right\rangle - \frac{2}{\sqrt{N}}\left|\phi\right\rangle
\end{aligned}$$

Now, apply D to the result of $U\left|s\right\rangle$:

$$\begin{aligned}
D\left(U\left|s\right\rangle\right) &= D\left|s\right\rangle - \frac{2}{\sqrt{N}} D\left|\phi\right\rangle \\
&= \left(2\left|s\right\rangle\left\langle s\right| - I\right)\left|s\right\rangle - \frac{2}{\sqrt{N}}\left(2\left|s\right\rangle\left\langle s\right| - I\right)\left|\phi\right\rangle \\
&= 2\left|s\right\rangle\left\langle s|s\right\rangle - \left|s\right\rangle - \frac{2}{\sqrt{N}}\left(2\left|s\right\rangle\left\langle s|\phi\right\rangle - \left|\phi\right\rangle\right) \\
&= 2\left|s\right\rangle - \left|s\right\rangle - \frac{2}{\sqrt{N}}\left(\frac{2}{\sqrt{N}}\left|s\right\rangle - \left|\phi\right\rangle\right) \\
&= \left|s\right\rangle - \frac{4}{N}\left|s\right\rangle + \frac{2}{\sqrt{N}}\left|\phi\right\rangle \\
&= \frac{N-4}{N}\left|s\right\rangle + \frac{2}{\sqrt{N}}\left|\phi\right\rangle
\end{aligned}$$

Thus completes one iteration of Grover's Algorithm. The register, which started in $\left|s\right\rangle$, an equal superposition of all states, is now slightly "less" of $\left|s\right\rangle$ and slightly "more" of $\left|\phi\right\rangle$. If we repeat the process, this interference would compound and the register would asymptotically approach $\left|\phi\right\rangle$. A simple geometric proof shows that the probability of measuring the register to be in state $\left|\phi\right\rangle$ after r repetitions is given by

$$P = \sin^2\left((2r+1)\arcsin\frac{1}{N}\right)$$

It turns out that $r = \frac{\pi}{4}\sqrt{N}$ maximizes this probability. Proof:

$$
\begin{aligned}
P(\phi) &= \sin^2\left((2\cdot\frac{\pi}{4}\sqrt{N}+1)\ \arcsin\left(\frac{1}{\sqrt{N}}\right)\right)\\
&\approx \sin^2\left(\left(\frac{\pi}{2}\sqrt{N}+1\right)\ \frac{1}{\sqrt{N}}\right)\\
&= \sin^2\left(\frac{\pi}{2}+\frac{1}{\sqrt{N}}\right)\approx 1
\end{aligned}
$$

Here the small-angle approximation is used to simplify the arcsin term, as the number of possible states of the register is much larger than 1 for reasonable n, so $1/N \ll 1$. Though you may have missed it, this is an incredible result! Recall that the classical algorithm to solve this problem required $O(N)$ iterations, while the number of iterations to give the optimal result grows with $\sqrt{N}$ for the quantum algorithm! To put it somewhat roughly, this quantum algorithm can search a group of N items while only actually looking at $\sqrt{N}$ of them.

4.2 Simon's Algorthim

Assume we have some oracle two-to-one function $f : \{0,1\}^n \to \{0,1\}^n$. There is a special bitstring $|\phi\rangle \in \{0,1\}^n$ such that, for some $|\psi_1\rangle \neq |\psi_2\rangle$, if $f(\psi_1) = f(\psi_2)$, then $|\psi_1\rangle = |\psi_2\rangle \oplus |\phi\rangle$. Note that $\oplus$ represents the bitwise XOR logic gate, and is equivalent to bitwise addition modulo 2. The problem is to find the value of $|\phi\rangle$.

To solve this problem, Simon's Algorithm uses a gate that operates on two bitstrings and returns two bitstrings:

$$U|x\rangle|y\rangle = |x\rangle|f(x)\oplus|y\rangle\rangle$$

The algorithm then proceeds as follows. First, both $|x\rangle$ and $|y\rangle$ are initialized to 0^n, and the Hadamard gate is applied to each bit of $|x\rangle$, giving the state

$$\frac{1}{2^{n/2}}\sum_{j=0}^{N-1}|j\rangle|0^n\rangle$$

Then, U is applied:

$$
\begin{aligned}
U|x\rangle|y\rangle &= \frac{1}{2^{n/2}}\sum_{j=0}^{N-1}U|j\rangle|0^n\rangle\\
&= \frac{1}{2^{n/2}}\sum_{j=0}^{N-1}|j\rangle|f(j)\oplus|0^n\rangle\rangle\\
&= \frac{1}{2^{n/2}}\sum_{j=0}^{N-1}|j\rangle|f(j)\rangle
\end{aligned}
$$

Finally, the Hadamard gate is again applied to each bit in $|x\rangle$, giving the state

$$\frac{1}{2^n}\sum_{j=0}^{N-1}\sum_{k=0}^{N-1}(-1)^{j\cdot k}|k\rangle|f(j)\rangle$$

$$= \sum_{k=0}^{N-1}\left(\frac{1}{2^n}\sum_{j=0}^{N-1}(-1)^{j\cdot k}|f(j)\rangle\right)|k\rangle$$

This is getting a little messy, but it will soon simplify. If we measured the state of the registers at this point, we would get a set of probabilities of measuring state $|k\rangle$:

$$P_k = \left\| \frac{1}{2^n} \sum_{j=0}^{N-1} (-1)^{j \cdot k} |f(j)\rangle \right\|^2$$

Because f is defined to be two-to-one, for a given value z in the range of f, there are two distinct values in f's domain that give z: $f(j_1) = f(j_2) = z, \quad |j_z^{(1)}\rangle \neq |j_z^{(2)}\rangle$. Instead of summing over inputs to f, we could also sum over outputs:

$$P_k = \left\| \frac{1}{2^n} \sum_z \left((-1)^{j_z^{(1)} \cdot k} + (-1)^{j_z^{(2)} \cdot k} \right) |z\rangle \right\|^2$$

Also by definition, $j_z^{(1)} = j_z^{(2)} \oplus \phi$, and vice versa. Using the identity $(a \oplus b) \cdot c = (a \cdot c) \oplus (b \cdot c)$:

$$\begin{aligned} P_k &= \left\| \frac{1}{2^n} \sum_z \left((-1)^{j_z^{(1)} \cdot k} + (-1)^{(j_z^{(1)} \oplus \phi) \cdot k} \right) |z\rangle \right\|^2 \\ &= \left\| \frac{1}{2^n} \sum_z \left((-1)^{j_z^{(1)} \cdot k} + (-1)^{(j_z^{(1)} \cdot k) \oplus (\phi \cdot k)} \right) |z\rangle \right\|^2 \\ &= \left\| \frac{1}{2^n} \sum_z (-1)^{j_z^{(1)} \cdot k} \left(1 + (-1)^{\phi \cdot k} \right) |z\rangle \right\|^2 \end{aligned}$$

Note that bitwise XOR is equivalent to addition modulo 2, and so we may factor out the common term $(-1)^{j_z^{(1)} \cdot k}$.

So, if $\phi \cdot k = 1$, then $P_k = 0$, and if $\phi \cdot k = 0$, then $P_k = 1/2^{n-1}$. This result implies that any measurement we make of a bitstring k will have the property $k \cdot \phi = 0$. Note also that that many k could satisfy this condition, and no k is more likely to be measured than any other k. This means that if we repeat this process many times, we can obtain a set of equations $k_i \cdot \phi = 0$ that can then be solved by a classical computer. The probability that some of the k_i are not linearly independent from others is nonzero, but repeated sampling can reduce this probability to arbitrarily close to zero. Thus, the time complexity of this algorithm, for any specified error probability $\epsilon > 0$ (a "bounded-error" algorithm), is $O(N)$. To compare, any bounded-error classical probabilistic algorithm solving this problem must have time complexity at least $O(2^{N/2})$ (Watrous, 2006). The quantum algorithm has an exponential time speedup over the classical equivalent.

4.3 Quantum Fourier Transform

The Quantum Fourier Transform is supposed to be the quantum analog of the classical Discrete Fourier Transform. Like all other quantum operators, it is unitary, and like a standard fourier transform, it transforms a ket in the standard basis to a ket in the frequency basis. However, we'll omit the details of how the transform works in favor of a qualitative approach, which will be sufficient to understand its use. The quantum Fourier transform, when acting on an arbitrary register state, gives

$$\hat{\mathcal{F}}\,|\psi\rangle = \hat{\mathcal{F}}\sum_{j=0}^{N-1} a_j\,|j\rangle = \sum_{k=0}^{N-1} a_k\,|k\rangle$$

where the resulting amplitudes a_k are the discrete fourier transform of the input amplitudes a_j:

$$a_k = \frac{1}{\sqrt{N}}\sum_{j=0}^{N-1} a_j e^{2\pi ijk/N}$$

This is useful for finding the period of a repeating number sequence, which by itself is a difficult task. The QFT is often used as part of a more complex algorithm and is rarely used by itself, as we'll see in the next section.

4.4 Shor's Algorithm

Most encryption on the internet is done with an encryption scheme called RSA. The scheme relies on the idea that multiplying two very large numbers can be done with relatively few computations, while factoring the product into the constituent multipliers can only be done very inefficiently. The best algorithms to do this on a classical computer run in something like $O(\exp[(\log n)^{\frac{1}{3}}(\log\log n)^{\frac{2}{3}}])$ (Odlyzko, 1995). The exponential nature of this complexity makes it completely impractical to try to factor large numbers on a classical computer: factoring a 250 digit number with these algorithms would take roughly 800,000 years, while a 1000 digit number would take 10^{25} years (Braunstein).

In contrast, Shor's algorithm can factor large integers with an exponential speedup over the classical method. Shor's algorithm is quite complex, and so here it is explained in english rather than with a rigorous mathematical derivation.

The idea goes as follows. To find a factor of some large number N, we first pick a random number x. The sequence given by $x^k \pmod{N}$ will be periodic, and we can find the period with the QFT algorithm. If the period is not even, continue choosing random x and computing the period until a period is even. Call the period r, and notice that $x^r = 1 \pmod{N}$. We can manipulate this like so:

$$x^r = 1 \pmod{N}$$

$$x^r - 1 = 0 \pmod{N}$$

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

This is equivalent to saying that the N is a divisor of the quantity $(x^{r/2}-1)(x^{r/2}+1)$, and so it must divide one (or both) of the two multiplied terms. Therefore, calculating the greatest common divisor between the terms and N will give a nontrivial divisor of N. A polynomial-time algorithm exists for calculating the GCD called Euclid's algorithm, and so the algorithm is finished with overall polynomial time complexity (Braunstein).

This version of Shor's algorithm looks much different than a rigorous derivation would, but the overall idea is the same: relating divisors to the period of a sequence is what makes factoring reasonable on a quantum computer. In fact, Shor's algorithm has time complexity $O((\log N)^2(\log\log N)(\log\log\log N))$, dramatically faster than an exponential time algorithm. It has also been implemented on a real quantum computer that factored 15 with 7 qubits (Vandersypen et al., 2001).

5 Discussion

The quantum algorithms presented here are highly specialized and took a lot of time to develop. It seems that the most likely use of a quantum processor would be to couple it with a classical processor and use it as an external computation unit, at least at first. Indeed, even though the computational complexity of a quantum algorithm can be exponentially less than the classical counterpart, this doesn't mean that a quantum computer will always solve the problem faster. Classical computers will not be completely replaced by quantum computers anytime soon.

A topic not covered here is quantum error correction. In practice, making qubits that don't decohere is difficult, so researchers have developed methods for correcting errors that result from decoherence and noise from outside the register. The actual mechanism is complex, involving entanglement of many qubits to employ redundancy.

Another application of quantum computing not covered in this paper is solving systems of linear equations with quantum algorithms. Algorithms exist that solve linear systems in polynomial of $\log(n)$ time, an exponential improvement over the classical analog (Harrow et al., 2009).

Finally, it is unknown if a quantum computer can solve all problems with faster time complexity than a classical computer can. Much of theoretical computer science is dedicated to studying different classes of problems and whether they can be solved in polynomial time or not. As we saw above, Grover's algorithm offers only a polynomial speedup, not an exponential speedup.

6 References

Braunstein. Quantum computation, http://www−users.cs.york.ac.uk/schmuel/comp/comp_best.pdf

Grover. A fast quantum mechanical algorithm for database search, 1999. arXiv:quant−ph/9605043v3

Harrow, Hassidim, and Lloyd. Quantum algorithm for linear systems of equations, 2009. arXiv:quant−ph/0811.3171v3

Odlyzko. The future of integer factorization , 1995. http://www.dtc.umn.edu/˜odlyzko/doc/future.of.factoring.pdf

Shor. Introduction to Quantum Algorithms, 2001. arXiv:quant−ph/0005003v2

Vandersypen et al. Experimental realization of Shor's quantum factoring algorithm using nuclear magnetic resonance, 2001. Nature, doi:10.1038/414883a.

Watrous. Lecture Notes, 2006. https://cs.uwaterloo.ca/˜watrous/CPSC519/LectureNotes/06.pdf