Quantum algorithms and complexity classes

Nathan Wu

December 14, 2022

1 Format

We first define a quantum Turing Machine(QTM). Then, we take a look at the corresponding complexity class, BPQ. Then, we will shift focus and explore some famous quantum algorithms

2 Assumptions

We assume knowledge of basic computation models up to and including the Turing Machine. We also assume knowledge of complexity classes.

3 Quantum Turing Machine

A quantum Turing machine(QTM) is defined similar to how a deterministic Turing machine is defined. That is, A set of finite states, an input and tape alphabet, a blank symbol, the start state, and the end state. The representation of a quantum Turing machine, however, is not universally agreed upon. However, all models adds a set of configurations, (q, T, i), where the machine is at a state q, the tape contains T, and the head points to the ith cell of the tape. This configuration is either embedded in the description of the transition function Δ via a transition matrix, or completely separate. However, in both cases, the QTM is modified as the configuration, according to some matrix/unitary U. Just as how P and NP stem from the languages a deterministic and nondeterministic Turing machine can accept, respectively, the complexity class that QTMs accept is called BQP. Quantum algorithms are inherently probabilistic, or not always guaranteed to produce a right answer. Similar to P or NP, langauges in BPQ can be decided by a QTM in polynomial time, and will correctly solve the problem $\frac{2}{3}$ of the time. Formally, a language L is in BQP if and only if there exists a QTM that accepts L with probability no less than $\frac{2}{3}$.

4 Quantum Algorithms

A quantum algorithm is an algorithm that can be performed by a QTM. That is, a set of instructions I that when performed on a QTM, computes a language in BQP. However, using a QTM does not allow for additional computational power; a QTM is equivalent to a standard Turing Machine in that all the langauges that can be computed by a QTM can also be computed by a TM. So what makes QTMs so interesting then, is the possible reduced computation time compared to a TM. Commonly, these algorithms are not actually performed on a QTM, but an equivalent model of computation, a quantum circuit.

4.1 Quantum Circuits

We won't formally define quantum circuits nor will we cover them extensively, but the following algorithms will all be implemented with quantum circuits so having some knowledge is requisite. Quantum circuits extend classical circuits in the same way QTMs extend the classical turing machines. Anything that can be done with the quantum circuit model can also be replicated with a QTM.
Like the classical bit, the qubit is used to store and communicate information. While the classical bits may be comprised of 0's and 1's, the qubit is a superposition of both. This is represented as 2-d vectors, $(\frac{0}{1})$ or $(\frac{1}{0})$, with the former describing an excited state describing the ground state of electrons. Then, two qubits can be represented as a 4-row vector, and so on. The principle of superposition, which states that if a quantum system has two states, then it can be in any linear combination of the two states, $a(\frac{0}{1}) + b(\frac{1}{0})$, as long as the norm of a and b add up to 1. The inbetweeness of this allows us to define the probability that the electron will enter either the excited state or ground state fully, which is the foundation for all quantum optimizations.

Another essential aspect of the quantum circuit model are gates. Gates are manipulations on qubits that transform them from one state to another. This is represented through square unitary matricies, and the transformation via matrix multiplication. A quantum gate that transforms n qubits is called a controlled gate and has dimensions 2^nx2^n. Here are a few gates. Identity: $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ NOT Gate: $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ SWAP Gate(swaps two qubits): $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

4.2 Shor's Algorithm

Perhaps the most famous example of a quantum algorithm is Shor's algorithm. Shor's algorithm is used for finding the prime factors of integers, and takes polynomial time on a QTM. Specifically, it takes $O(logN)$ time to factor N, which is exponentially faster than the known classical method to factor. Shor's algorithm is composed of two parts, a part that can be performed on a classical

turing machine and a part that utilizes a QTM. We will first show a classical implementation of Shor's algorithm, then show where a QTM can improve the runtime.

4.2.1 Classical Implementation

Let's say we want to factor N. Then we choose some k between 1 and N and we find $gcd(k, N)$. This can be done via the Euclidian algorithm. If the gcd is not 1, then whatever the gcd is is a factor of N. However, if it is 1, then we want to find $ord_N k$, or the smallest positive integer p such that $k^p \equiv 1$ mod N. This is because then $k^p - 1 \equiv 0$ mod N and is then a factor of N. Finding the order can be accomplished through testing factors of $\phi(N)$ and testing using binary exponentiation. Now that we have a value of p, we perform a few checks. If it is odd, we go back to finding a different value of k, and if it is even, we continue. Else, if either $a^{\frac{p}{2}} - 1$ or $a^{\frac{p}{2}} + 1$ divides N, we continue. If all of these checks pass, a factor of N will be $gcd(a^{\frac{p}{2}} - 1, N)$.

4.2.2 Quantum Optimizations

Most of this process can be performed quite efficiently on a classical turing machine because of the euclidean algorithm and binary exponentiation. However, the process of finding $ord_N k$ can be optimized with a QTM with quantum Fourier transform. Like the discrete Fourier transform, the quantum Fourier transform helps approximate periodic functions, which can be utilized if we define $f(x) = a^x$ and try to find p such that $f(x) = f(x + p)$.

4.2.3 Quantum Fourier Transform

Like the classical Fourier transform, the quantum Fourier transform decomposes the function into quantum states and then into a superposition of sinusoidal waves at different frequencies. Essentially, the quantum Fourier transform provides a way of expressing a quantum state as a superposition of sinusoidal waves at different frequencies.

4.3 Grover's algorithm

Grover's algorithm provides a faster method to search an unsorted and unstructured database. Normally, this search will take $O(n)$ time to find a certain element, but Grover's algorithm speeds this up quadratically to provide a runtime of $O(\sqrt{N})$. This runtime has been proved (even before Grover's algorithm was discovered) to be the lowest runtime possible for searching on an unstructured database. Grover's algorithm makes use of the concept of quantum superposition, a quantum turing machine can be in multiple states simultaneously, allowing it to check multiple elements.

Grover's algorithm utilizes a quantum oracle, which is essentially a unitary matrix that applies an operation to a given quantum state and transforms that state. In Grover's algorithm's case, the oracle is used to encode information about desired element in the database. The oracle is then applied multiple times to the tape along with other quantum operations in order to zero in on the position of the desired element.

4.3.1 Amplitude Amplification Trick

The amplitude amplification trick is a technique used in quantum computing to amplify the amplitude (or probability) of a desired quantum state. It is the bread and butter of how QTMs are able to speed up runtimes of certain tasks by such a large margin. This was shown in Grover's algorithm by the usage of an oracle to gain more information about the desired element and narrow down its position. In general this means manipulating the amplitudes of the different quantum states in a quantum system, such as creation of a superposition of all possible states, or the application of a quantum measurement that "collapses" the superposition into a single state.

4.4 Deutsch-Jozsa algorithm

The Deutsch-Jozsa algorithm was specifically designed to show a difference between quantum classical complexity classes. The problem is as follows:

Given an oracle that implements $f : 0, 1^n \rightarrow 0, 1$, and that the function is either constant(either 0 or 1 on all outputs) or balanced(half of the outputs are 0s), determine whether the function is balanced or constant.

4.4.1 Classical Solution

For a classical turing machine or another classical system of computation, the runtime is exponential. $2^{n-1} + 1$ comparisons are necessary to check if a string is balanced if the same output is given for every input tried. Since there are 2^n possible strings, we need to check 1 more in order to determine if it is balanced or constant.

4.4.2 Quantum Solution

Using an oracle, this problem can be solved with a single query. The oracle used here is the same unitary matrix as the one used in Grover's algorithm. Taking advantage of the amplitude amplification trick, Deutsch-Joza's solution is able to apply this oracle once and search the space completely for any differing outputs.

5 Bibliography

5.1 References

Vazirani, Umesh V. "Chapter 10: Network Flow Algorithms." Algorithms, Berkeley, CA: University of California, EECS Department, 2011, people.eecs.berkeley.edu/ vazirani/algorithms

Deutsch-Jozsa Algorithm. (n.d.). Retrieved from https://qiskit.org/textbook/ch-algorithms/deutsch-jozsa.html

Deutsch-Jozsa Algorithm. (n.d.). Retrieved from https://quantum-computing.ibm.com/composer/docs/iqx/guide/deutsch-jozsa-algorithm