Quantum Computation

quinten tupker

October 8 2020 - October 15, 2020

Introduction

These notes are based on the course lectured by Professor Richard Jozsca in Michaelmas 2020. Due to the measures taken in the UK to limit the spread of Covid-19, these lectures were delivered online. These are not meant to be an accurate representation of what was lectures. They solely represent a mix of what I thought was the most important part of the course, mixed in with many (many) personal remarks, comments and digressions... Of course, any corrections/comments are appreciated.

This course is meant to be a second course on quantum computation. In particular, all the prerequisite knowledge is covered in Cambridge's Part II Quantum Information and Computation course. Lecture notes for this course can be found online.

Now, to start describing course content. Quantum Computation studies algorithms that can be run on quantum computers. Although they have yet to be implemented in practice (but are definitely being developed at a rapid pace), and in particular how they differ with classical computation. A remarkable result, is that at least superficially, quantum algorithms appear to be more powerful than classical algorithms, although it remains unclear if this is definite fact, or if it simply easier for humans to solve complex problems using quantum algorithms instead of classical algorithms. This is remarkable in many ways in a philosophical sense, but here we focus on how to take advantage of these changes. As such, we will begin with a review and extension of one of the most famous quantum algorithms, which is Schur's factoring algorithm.

1 Schur's Algorithm Revisited and the Hidden Hidden Subgroup Problem

Schur's factoring algorithm finds a factor for an arbitrary number N. It does not perform a complete factorisation. It merely computes a factor, which of course can be repeated arbitrarily to get a complete factorisation, but that is not the point here. The complexity of such an algorithm is typically measured in terms of the number of digits of N, which we may denote $n = \ln(N)$. In these

terms Schur's algorithm is $O(n^3)$, which means it is "efficient" (computationally feasible) or

Definition 1 (efficient algorithm)**.** An algorithm is efficient if it runs in polynomial time, which generally means it is considered doable in practice.

By comparison, the fastest known classical algorithm runs in $O(e^{n^{1/3}\ln(n)^{1/3}})$. Anyways, here is an outline of Schur's algorithm:

1. choose $a < N$ such that $(a, N) = 1$ (coprime). This can be done efficiently, since the probability of a being coprime is fixed, and we can quickly calculate the GCD using Euclid's algorithm. Then consider $f(x) = a^x (\mathrm{mod} N)$.
2. Use quantum algorithms to calculate the period of the this function (so we have converted a factoring problem into period finding). Since a is coprime, it is guaranteed to be periodic.
3. compute the factor using number theory

The crucial component here is the period finding, which cannot be done efficiently using a classical algorithm. So let's review quantum period finding. Also, note that as usual, quantum oracles are implemented as unitary operators by converting $f : \mathbb{Z}_M \to \mathbb{Z}_N$ to $U_f |x\rangle |0\rangle \mapsto |x\rangle |f(x)\rangle$. Then, if f has period r (unknown), and f is one-to-one on every period, then period finding can be done using

1. make $\frac{1}{\sqrt{M}} \sum_0^{M-1} |i\rangle |0\rangle$
2. apply U_f
3. measure the output register to get

$$\frac{1}{\sqrt{A}}(|x_0\rangle + |x_0 + r\rangle + \cdots + |x_0 + (A-1)r\rangle) |f(x_0)\rangle$$

Now the next step is the tricky part, and really is what uses the "quantum magic" here, and that involves the use of the Quantum Fourier Transform (QFT). [This ends lecture 1]

4. We then apply the QFT, which maps $|k\rangle \mapsto \sum_{y=0}^{M} e^{xy} |y\rangle$, which, after some calculation (use $\sum e^{2\pi k x/y} = y\delta_{xy}$) leaves us with

$$\mathrm{QFT} |\mathrm{per}\rangle = \sqrt{A/M} \sum_{k=0}^{r-1} \omega^{x_0 k M/r} |kM/r\rangle$$

5. Making a measurement we get $C = k_0 M/r$, so $\frac{k_0}{r} = \frac{C}{M}$. If k_0, r are coprime, we are done, since we can reduce $\frac{C}{M}$ to simplest terms (use Euclid's algorithm to cancel out the gcd). Now, number theory tells

us that the probability of being coprime is finite and shrinks slowly (as $O(1/\log\log(M))$), and so we can just repeat until we get the right period. Since f is one-to-one on each period, it is easy to check if our period is correct.

I feel that just being able to check if the period is correct is a somewhat lame reason to require that the function be one-to-one on each period, but improvements although not difficult, would complicate this explanation.

Anyways, let's see if we can motivate the Quantum Fourier Transform a bit better. The challenge we face is that our state, $|R\rangle$ takes the form

$$|R\rangle = \sum_k a_k |x_0 + kr\rangle$$

for an arbitrary x_0. In other words, we have an arbitrary shift that we want to ignore some how. How do we do that? A natural way to spot "things that ignore shifts" would be to define the shhift operator

$$U|x\rangle = |x + 1 \bmod M\rangle$$

and then, how do we say, "we don't care about " U? We look for the eigenvectors of U, which are by definition, the states least affected by U. Fortunately, U is a permuatation matrix, so unitary, so is a quantum gate. Then, the eigenbasis of U is what we may call the set of shift-invariant states χ_k. If we write R in terms of this basis we are bound to get a state of the form

$$\sum_k a_k \lambda_k^{x_0} |\chi_k\rangle$$

where $\mathbb{P}(k) = |a_k \lambda_k^{x_0}|^2 = |a_k|^2$ since as eigenvalues of a unitary matrix, $|\lambda_k| = 1$ always. So as expected, probabilities are preserved (this is not that crucial - but it's important they don't differ that much). Anyways, important is that this transformation allows us to express our state as a sum of multiples of the period, which is what we want.

All that remains is to find this basis, and the operation that expresses them in terms of it. As eigenvectors of a unitary matrix are orthogonal, all we need to do is zip the eigenvectors into a matrix. These eigenvectors are just of the form $e^{2\pi ikl/M}$, so we get the Quantum Fourier Transform we expect. [End of lecture 2]

1.1 The Hidden Subgroup Problem (HSP)

The hidden subgroup problem asks the question how can we find subgroup K of group G (this course only considers finite G) given a function $f : G \to X$ that is an invertible function of the left cosets of K. Our goal is to solve this problem in $O(poly(\ln(|G|)))$.

Examples of this include Schur's algorithm where we have $G = \mathbb{Z}_p^*$, then $K = 0, r, 2r, \ldots$, and then f used before works for our purpose. Another example is calculating **discrete logarithm**, which if done efficiently could break

encryption methods. This involves calculating logarithms on $\mathbb{Z}_p^*$, so given x finding y such that for group generator g, $x = g^y$. To formulate this as an HSP consider

$$\begin{aligned} f : \mathbb{Z}_{p-1} \times \mathbb{Z}_{p-1} &\to \mathbb{Z}_p^* \\ (a, b) &\mapsto g^a x^{-b} = g^{a-yb} \end{aligned}$$

where we can see that $f(a_1, b_1) = f(a_2, b_2)$ iff $(a_2.b_2) - (a_1, b_1) = \lambda(y, 1)$ some λ. So using $K = \{\lambda(y, 1) | \lambda \in \mathbb{Z}_{p-1}\}$ works.

Those both involve abelian groups, but a big area of interest is solving the problem for non-abelian groups. Examples of such problems include finding the Automorphism group of a graph. For graph A this means finding $\mathrm{Aut}(A)$ a subgroup of the group of permutations of the vertices of A such that the overall structure (edges) are preserved. This can be formulated quite naturally into an HSP by taking X to be the set of all n vertex graphs, and then to consider the function $f_A(\pi)$ which applies permutation π to A. Clearly the result depends only on the coset of $\mathrm{Aut}(A)$.

Even more famous is the **Graph Isomorphism** problem (GI), which is to check whether or not two labelled graphs are isomorphic (so whether there exists a permutation turning one into the other). This can be converted into an HSP, although the process is more complicated. There also exists few good classical algorithms, although in 2017 someone found an algorithm doing it in quasi-polynomial time.

But how far will we get with quantum algorithms. The abelian case has been fully solved, but unfortunately the non-abelian case (which the above two problems belong to) remains an important unsolved problem in the field. [End of lecture 3]