Quantum Computation

quinten tupker

October 8 2020 - October 8, 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.

0.1 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]