Quantum Computing Project

Group Report

Max Barnett Lizzie Dobson Josh Dykstra
Nicoline Hemme Asuka Nakamaru-Pinder Lewis Trainer

February 10, 2020

Contents

1 Introduction and Overview

1.1 Motivation for Quantum Computing

A qubit is a two-state quantum system which can encode one bit of information about a system. Unlike classical bits, qubits can exist in a superpostion, and multiple qubits can become entangled with each other. Two big advantages of quantum computing over classical computing, are that quantum mechanics naturally allow for many reversible operations, and that a quantum system can be in several states at once and applying operations to a qubit will perform the operations in many states at once.

Quantum computers are, with today's knowledge and technology, difficult to build to be efficient and cheap enough for most purposes. However, it has huge potential when certain barriers can be overcome and therefore it remains an interesting subject to study and develop. Consider the Gottesman-Knill Theorem[1];

> Suppose a quantum compuation is performed which involves only the following elements: state preparations in the computational basis, Hadamard gates, phase gates, controlled-NOT gates, Pauli gates, and measurements of observables in the Pauli group (which includes measurement in the computational basis as a special case), together with the possibility of classical control conditioned on the outcome of such measurements. Such a computation may be efficiently simulated on a classical computer.

The theorem will not be proven here, but it states that a (limited) simulation of a quantum computer can be executed on a classical computer, which will allow for better understanding and open up for further exploration. This report will introduce, present and discuss our own quantum computer simulator.

1.2 Background

1.2.1 A Brief History of the Subject

Quantum computing was the startling result of the metaphorical collision of quantum mechanics and information theory, as such, defining its history is complicated. Timelines for quantum computing tend to vary on the basis of their creator's perspective. For the purposes of this report, some history of the contributory disciplines will be discussed to add context.

Quantum mechanics has a long and interesting history (far too much for a section titled 'brief history'), but 'modern' quantum mechanics, the quantum mechanics with which we are familiar today, can be traced most clearly to the first half of the 20th century.

During this period many of the greatest minds in modern Physics began to establish what would become the foundations of the discipline. The early stages of quantum mechanics came about as a result of scientific observations that were not consistent with classical physics. Two of the early pivotal figures in this nascent development of quantum theory were Max Planck and Albert Einstein. In 1900 Planck derived the law of black-body radiation which brought about the ideas of quantisation, ideas that were incompatible with classical physics [2]. Soon after, in 1905 Einstein built upon this idea and released a paper titled *On a Heuristic Viewpoint Concerning the Production and Transformation of Light*. Einstein's publishing assumed that photonic energy emissions and absorption could only occur in discrete amounts, what he called "quanta"[3]. As the magnitude of this paradigm shift within physics began to be realised, quantum theory began to gain momentum.

A significant sign of the progress and change in approach was the wide adoption of Niels Bohr's Copenhagen Interpretation (the idea that the properties of physical systems are not definite until observed, thus quantum mechanics relies on prediction through probability). This tide of 'new' physics included Erwin Schrödinger's rather famous equations and moved into further unification and formalisation as the decades progressed quickly becoming what we know it as today.

As quantum mechanics continued to be formalised and advanced, another revolutionary development occurred in a seemingly vastly separate area of research. In 1948 a mathematical research paper birthed the discipline of information theory. *A Mathematical Theory of Communication* by Claude E. Shannon is considered the originative text that lead the establishment of information theory, bringing about the analysis of statistical processes as both a qualitative and a quantitative model. Once information theory was formally established the quantitative ideas of information and other core tenets of information theory as a whole rapidly evolved. One large factor in this advancement was that much previous research across scientific fields could now be either categorised under information theory or used to develop it (much of the mathematics behind information theory is based on thermodynamic research by L. Boltzmann and by J. Willard Gibbs).

Finally, as research in quantum mechanics and information theory both continued to accelerate a startling realisation struck. The preface N. David Mermin's *Quantum Computer Science: An Introduction* sums it up beautifully with the following:

> *"It was almost three quarters of a century after the discovery of quantum mechanics, and half a century after the birth of information theory and the arrival of large-scale digital computation, that people finally realized that quantum physics profoundly alters the character of information processing and digital computation. For physicists this development offers an exquisitely different way of using and thinking about the quantum theory. For computer scientists it presents a surprising demonstration that the abstract structure of computation cannot be divorced from the physics governing the instrument that performs the computation."*[4]

Each decade the number of significant advancements within the field of quantum computing seems to grow exponentially. Ironically, reading any thoroughly researched

quantum computing timeline almost feels as though observing a version of Moore's Law (number of transistors in a dense integrated circuit doubles about every two years) in action. As such, to keep the history brief, it seems prudent to describe the advancements with broader strokes across the decades.

The 1960s yielded what many view to be the first definitive event in quantum computing. Stephen Weisner, a physicist, created a cryptographic tool known as conjugate coding which was an application for what would come to be known as quantum programming[5]. Further research in the 1970s began to show the incompatibility of quantum computing with classical systems as well as early attempts a the creation of a quantum information theory [6].

The 1980s marked the transition from quantum information theory to quantum computing as well as the recognition of the field and the first glimpses of its potential. Perhaps the most important event in the early history of quantum computing, was Paul Beinhoff's description quantum mechanical Hamiltonian models of computers[7]. Beinhoff finally supplied an idea of what quantum computing might actually look. Beinhoff and others continued to drive research yielding further theoretical modelling of quantum computing, the introduction of gates (operators that act transformatively on qubits), and even the first description of the universal quantum computer.[8]

As we enter the last 3 decades, the vast quantity of research and dramatic advancement in quantum computing becomes difficult to document with broad strokes. The 1990s brought forth a vast array of quantum computing research including development of the framework in which a quantum computer could one day be made. Of particular note to this research paper is the discovery of 2 important quantum algorithms; Shor's algorithm (1994) for integer factorisation and Grover's quantum database search algorithm (1996).[9] 1998 included a landmark event in which the first experimental demonstration of a quantum algorithm ran on a working 2-qubit quantum computer (quickly followed by a demonstration of Grover's algorithm). The aforementioned events are just a few among an array of incredible work done in the field of quantum computing.

The turn of the 21st century and the early 2000s easily picked up where the last decade left off, with increasingly large experimental systems, the publication of a standard textbook, and the beginnings of quantum error correction. Unfortunately, around 2005 the speed of research grew to be so great that single years match previous decades in terms of output. As such, historical discussion going forward will focus purely on events deemed exceptionally noteworthy or of particular relevance to this report. Omission from this report is in no way an indication of the importance of an event but rather a pragmatic decision.

A noteworthy event near the end of 2005 was the creation of the first quantum byte, or *qbyte*.[10] The size of experimental devices continued to increase in 2006 as did progress toward the creation of the first quantum gate. In the same year, the University of Copenhagen managed to develop quantum teleportation between photons and atoms (a method of quantum information transfer which has dramatic theoretical impact).[11]

2007 was a particularly active year, among the many steps taken, one stands out particularly, the use of photonic quantum computing by two independent labs to factor numbers.[12] The use of quantum computing to factor numbers is a very important

development for the field as a whole. While on the surface the factorisation seems a small result, one of the huge potential use cases for quantum computing lies in its effect on cryptography and information transfer as a whole. Prime numbers and factoring systems are at the core of modern encryption and information infrastructure, everything from basic password protection and secure internet connections to classified government secrets use techniques based on these tenets. Quantum computing is quite literally a game changer offering the ability to factor at dramatically faster speeds among other developments, the ramifications of such developments are enormous and remain partially unclear (such is the size of the impact).

In the following years leading up to the new decade systems continued to be advanced and refined. The growing realisation of quantum computing's potential lead to large investment and growth in competition as companies like Google, Microsoft, and IBM started to work alongside academic institutions.

The 2010s much like the previous decade represented an explosive increase not only in interest and activity within the field, but in progress. The latter half of the 2010s falls under the current state of the field and will be discussed in the following section. For the first half of the decade, rather than dissect each year it should suffice for our purposes to look briefly at a few of the noteworthy breakthroughs.

Computers, classical or otherwise are at their core a series of transistors. A huge advancement to the possibility of high level quantum computers came in 2012 when a group of physicists managed to create a transistor from a single atom.[13] This serves as an important event because, the single atom transistor would serve as a base unit within the currently postulated architecture of a quantum computer.

These days any thought of computers almost invariably includes thought of the internet. In the modern collective conscience computers and the internet are virtually inseparable. A huge shape on the horizon of quantum computing is the creation of a quantum internet. With the advent of successful quantum computing, the internet as it is now, would not survive. The current infrastructure is not 'quantum proof', that is to say its total reliance on cryptographic infrastructure (RSA code security which relies on the difficulty of number factoring) that is vulnerable to quantum computing means a whole new system would need to be put in place. This revelation should serve to highlight the importance of the successful transfer of data via quantum teleportation over 10 feet with a zero percent error rate. Quantum teleportation is a method of information transfer that will be the core of any quantum internet. The successful test in 2014 marked a huge step toward the realisation of said ambition.

1.2.2 Current state of the field

The current state of quantum computing is exciting, confusing and unique. While there are plenty of resources and indeed ambition being injected into the field, there has been no commercially viable implementations of any genuine quantum algorithms on the *"noisy"* quantum computers currently available. As a result of this, quantum error correction is a prolific field of research. Quantum computing as a discipline is rapidly growing and appears on the cusp of breaking through to wide spread commercial use and the

application of quantum computing principles across the board. Quantum computing systems are constantly growing larger and being used to perform continuously more complicated operations. Last year a 72 qubit chip was created by Google[14], and this year IBM launched *IBM Q System One*, the first commercial quantum computer.[15]

1.2.3 Limitations and how they might be overcome

As exciting a field as quantum computing is, there are multiple issues which still serve as limitations to the field and must be resolved in order to achieve the vast potential suggested.

As this is a physics based report the limitations being addressed will be investigated from this perspective. There are several other limitations such as issues with complexity that are better suited to discussion within mathematics and informatics.

The limitations to quantum computing can be divided into 2 categories, conceptual and physical. The conceptual limits surround the actual applications of the methods designed for use on quantum systems. Grover's search algorithm on the surface seems an incredible jump in ability, however, this is within an idealised system. In reality, there are multiple factors which offset the effectiveness of Grover's and other algorithms. Using Grover's as an example seems prudent since it serves as a focus of this report, but these issues and similar limitations plague the different applications of quantum computing.

Removing Grover's from the ideal system reveals that while it is highly efficient when the set of values is unknown it struggles with more defined bases. Research has shown that in effectiveness is confined to implicit databases.[16] Overcoming this problem is not really an option as it does not have to do with implementation. Rather, it is a limitation which can be acknowledged and circumvented by applying the algorithms and methods which best suit a given situation. This should serve as a slight check on the 'hype' surrounding these different new quantum algorithms. In this same trend, it is important to compare quantum algorithms with their actual 'competitors', the specialised algorithms designed and tuned to specific problems, rather than the basic search methods.

Another result of removing Grover's from the ideal system is to note cases where it functions almost as slowly as classical search methods. Grover's algorithm relies on the fact it requires fewer queries than classical search methods. However, if the time required per query is significantly large Grover's will function at speeds comparable to basic classical algorithms.[16]

There are many quantum algorithms and each face different but similar problems to Grover's, these are the conceptual issues that serve as hurdles in the progress of the field. The second aforementioned category of limitations and perhaps the larger obstacle, is physical, the issues of hardware.

While the mathematical issues may cause application problems, they could still be utilised in specific situations if the hardware were sufficient. What has stood in the way of effective creation of quantum computers has been the inability to physically make the systems. Quantum computing has dramatically advanced, but for all its potential, it has yet to be demonstrated at full scale.

One of the most obvious issues is that the current processes to build these systems are

prohibitively expensive. While that is a large issue it is also an easily solved one through further investment meaning larger coffers and also leading to a reduction in costs over time. The larger hardware problem is an issue known as *quantum decoherence*, or more commonly quantum noise. This problem is based in the inherent quantum nature of qubits. One of the basic principles of quantum mechanics is uncertainty, the uncertainty of qubits presents a problem when trying to use them as parts of a computational system. This noise has been observed to destroy data stored in qubits. [17] The comparable problem in a classical system would be if the piece of your computer's drive that held a certain piece of information simply disappeared. One of the reasons quantum noise is such an issue is that there is no way to overcome the problem of inherent uncertainty, even in a perfect system, the nature of the qubits causes the problem.

Inherent uncertainty is not the only hardware issue facing quantum computing. Much more practically is the actual difficulty in building a perfect system. Uncertainty due to an imperfect system compounds on top of the aforementioned inherent uncertainty. Unlike inherent quantum uncertainty, imperfection can theoretically be overcome, but doing so is very difficult.

Quantum computing is not without its limits and there are many challenges still to overcome before it becomes fully viable. However, there are huge advancements that have and continue to be made and the potential of quantum computing in the future is vast.

1.3 Notation for Quantum States

A qubit $|\psi\rangle$ is a vector in $\mathbb{C}^2$ of unit length. The computational basis[1] for a single qubit is the set of base states,

$$\left\{ |0\rangle \equiv \begin{bmatrix} 1 \\ 0 \end{bmatrix} \ , \quad |1\rangle \equiv \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right\}. \tag{1.1}$$

A generic state $|\psi\rangle$ can then be represented as $|\psi\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle$ where $|\alpha_0|^2 + |\alpha_1|^2 = 1$. As unit vectors in $\mathbb{C}^2$, qubits can either be thought of as living in S^3, or, if we ignore the overall phase, in S^2. For the latter, writing a state as $|\psi\rangle = \cos(\theta/2)\,|0\rangle + e^{i\phi}\sin(\theta/2)\,|1\rangle$ parametrises a unit sphere, known as the *Bloch sphere* (Figure 1.1), with θ and ϕ as the polar and azimuthal angles. An operator $\hat{O} \in \mathrm{U}(2)$ acting on a single qubit can then be thought of as a rotation on this sphere.

A quantum register is a collection of n qubits, with states

$$|\Psi^{(n)}\rangle = |\psi\rangle_n \otimes |\psi\rangle_{n-1} \otimes \cdots \otimes |\psi\rangle_1 \in \mathbb{C}^{2n}, \tag{1.2}$$

where the so-called 'little endian' notation means that we start with the first qubit $|\psi\rangle_1$ on the right, and successively take tensor products with $|\psi\rangle_2$, $|\psi\rangle_3$, etc.

[1] An alternative basis for the qubit states is denoted by $\{|+\rangle, |-\rangle\}$, where

$$\left\{ |+\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}} \equiv \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \ , \quad |-\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}} \equiv \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ -1 \end{bmatrix} \right\}.$$

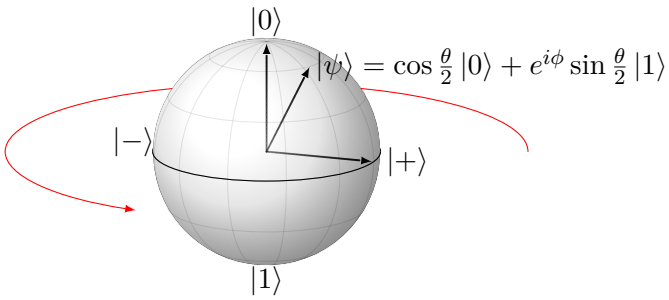


Figure 1.1: The north and south poles of the Bloch sphere correspond to the states $|0\rangle$ and $|1\rangle$ respectively, while states on the equator correspond to those with equal probabilitity of being observed in either.

We use the shorthand $|\psi_n \dots \psi_1\rangle \equiv |\psi_n\rangle \otimes \cdots \otimes |\psi_1\rangle$ so that the computational basis for the register of length n is:

$$\left\{ |0\rangle \equiv \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, |1\rangle \equiv \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}, \dots, |2^n - 1\rangle \equiv \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix} \right\} . \tag{1.3}$$

This ordering of qubits means that we write the state $|x\rangle$ (where $x \in \{0, 1, \dots, 2^n - 1\}$) as

$$|x\rangle \equiv |x_{n-1}, x_{n-2}, \dots, x_2, x_1, x_0\rangle \equiv |\sum_{j=0}^{n-1} 2^j x_j\rangle = \bigotimes_{j=0}^{n} |x_j\rangle \ , \tag{1.4}$$

where the tensor product is understood so that the kets are ordered with decreasing j. For example, the state '$|3\rangle$' in a register of four qubits would be represented in the computational basis as:

$$|3\rangle \equiv |0011\rangle \equiv |0\rangle\otimes|0\rangle\otimes|1\rangle\otimes|1\rangle \equiv \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}^T . \tag{1.5}$$

Note that in general, a state in the register does not correspond to the tensor product of n individual qubit states unless there is no entanglement.[2]

A general state in the register is a unit vector in $\mathbb{C}^{2^n}$, i.e. it can be thought of as living on the $(2^{n+1} - 1)$-sphere. Rotations on this sphere are generated by operators in $\mathrm{U}(2^n)$ called *gates*.

1.4 Gates in Quantum Computing

A *gate* is an operator that takes in k qubits and acts on them via a unitary — and hence invertible — transformation to produce k output qubits.

[2]Consider a pair v and w which are elements of the vector spaces V and W respectively. Whilst we can associate $v \otimes w$ to a state in the space $V \otimes W$, the converse is not true.

1.4.1 Single-qubit gates

The trivial example of a gate is the (2×2) identity matrix which leaves the qubit unchanged. Other gates which act on single qubits include the Pauli gates (X, Y, Z) and the phase-shift gate R_ϕ, which have the following representations in the computational basis:

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, \quad Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, \quad R_\phi = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\phi} \end{bmatrix}. \tag{1.6}$$

In the Bloch sphere picture, the effect of R_ϕ is clearly to rotate anticlockwise by an angle ϕ about the $\hat{\mathbf{z}}$ axis. The 'spin-flip' gate $Z = R_\pi$ then rotates by π, while the X and Y gates map the point (θ, ϕ) to $(\pi - \theta, -\phi)$ and $(-\theta, -\phi)$ respectively. Since any SU(2) operator can be represented by a combination of the Pauli matrices (X, Y, Z) then these (together with the identity) form a complete basis of single-qubit gates up to an overall phase factor. In practice, other gates are often useful. One particularly important example is the *Hadamard* gate, H_d, which takes the $|0\rangle$ state to an equal superposition of $|0\rangle$ and $|1\rangle$ states:

$$H_d = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \qquad H_d |0\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}}. \tag{1.7}$$

Taking tensor products of these single-qubit gates produces operators that act on a multi-qubit register. As an example, suppose we had 3 qubits ψ_0, ψ_1 and ψ_2 and we want to act on ψ_0 with the Hadamard gate, ψ_2 with the X gate, and leave ψ_1 unchanged. This would correspond to acting on the register with the operator $X \otimes I \otimes H$, since

$$(X \otimes I \otimes H)(\, |\psi_2\rangle \otimes |\psi_1\rangle \otimes |\psi_0\rangle \,) = (X |\psi_2\rangle) \otimes (I |\psi_1\rangle) \otimes (H |\psi_0\rangle). \tag{1.8}$$

In the following, we use the shorthand '$^{\otimes n}$' to denote taking the tensor product of n copies of an operator or state. For example, to initialise a register of n qubits in the zero state to an equal superposition, we act with $H_d^{\otimes n}$:

$$H_d^{\otimes n} |0\rangle = (\bigotimes_n H_d |0\rangle_{\text{single-qubit}}) = \frac{1}{2^{n/2}} \sum_{k=0}^{2^n - 1} |k\rangle \,. \tag{1.9}$$

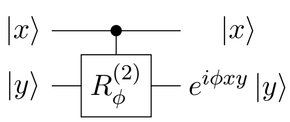
1.4.2 Multi-qubit gates

A useful gate which acts on a pair of qubits and switches them is the 'SWAP' gate,

$$S = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{represented by} \quad \begin{matrix} |x\rangle \\ |y\rangle \end{matrix} \!\!\begin{matrix} \text{—}\times\text{—} |y\rangle \\ \text{—}\times\text{—} |x\rangle \end{matrix} \tag{1.10}$$

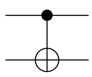
Another useful gate is the controlled phase-shift gate $R^{(2)}_{\phi}$ and its special case, the controlled-V (or c-V) gate:

$$R^{(2)}_{\phi} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\phi} \end{bmatrix}$$

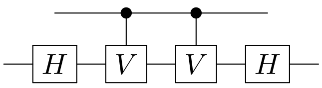


$$\text{and} \quad V = R^{(2)}_{\pi/2} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & i \end{bmatrix}. \tag{1.11}$$

Note that when working out how the gate will act on the register, linearity means that we only need consider the effect on the states in the computational basis. From V and H, it's possible to construct new gates that act on pairs of qubits, such as the c-NOT gate,

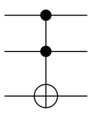


$\equiv$

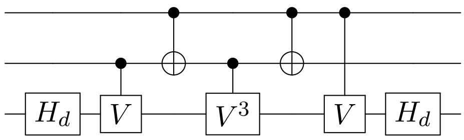


$$= (I \otimes H)V^2(I \otimes H) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \tag{1.12}$$

or on triples of qubits, such as the c^2-NOT gate or Toffoli Gate:



$\equiv$



$$= \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}. \tag{1.13}$$

This can of course be generalised to arbitrary numbers of input and output qubits, and it can in fact be shown that *any* n-qubit gate can be constructed from $\mathcal{O}(4^n n)$ V and H gates, to an arbitrary degree of precision.

There are many other sets of gates for which this is possible; these are called *universal sets* of quantum gates. By the *Solovay-Kitaev theorem*, then any set of single-qubit gates which form a dense subset of SU(2) has this property[3]. A more precise statement of the theorem and an algorithm for finding such universal sets is explained in detail in [18].

1.5 Some Useful Theorems and Rules

1.5.1 The *No-cloning* Theorem

The *No-Cloning* theorem says that it is not possible to create a linear operator that will perfectly duplicate a state $|\psi\rangle$, that is, there is no U such that $U(|\psi\rangle\,|0\rangle) = |\psi\rangle\,|\psi\rangle$. A

[3]Here we consider SU(2) rather than U(2) because the overall phase factor is not physically meaningful

simple proof of this is given in Mermin's book, [4], using linearity to show that:

$$\begin{gathered} U(|\psi\rangle\,|0\rangle) = |\psi\rangle\,|\psi\rangle\ , \qquad U(|\phi\rangle\,|0\rangle) = |\phi\rangle\,|\phi\rangle \\ \implies U((\alpha\,|\psi\rangle + \beta\,|\phi\rangle)\,|0\rangle) = \alpha\,|\psi\rangle\,|\psi\rangle + \beta\,|\phi\rangle\,|\phi\rangle \\ \text{but} \quad U((\alpha\,|\psi\rangle + \beta\,|\phi\rangle)\,|0\rangle) = (\alpha\,|\psi\rangle + \beta\,|\phi\rangle)(\alpha\,|\psi\rangle + \beta\,|\phi\rangle \implies \alpha = \beta = 0\,. \end{gathered} \tag{1.14}$$

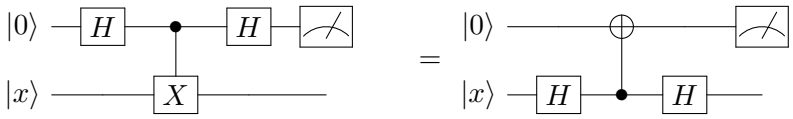
This is physically essential as otherwise one could create an arbitrary number of qubits in identical states and measure each one to gain more information about the qubit.

1.5.2 The Generalised Born Rule

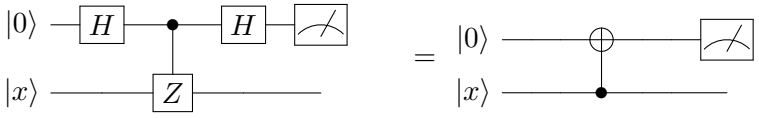
The generalised Born Rule states that when measuring a qubit, regardless of whether it is part of a higher qubit state, the measurement will reveal the qubit to be in either the $|0\rangle$- or $|1\rangle$-state, with probability $|\alpha|^2$ or $|\beta|^2$, respectively.[4] Hereafter the superposition will have collapsed.

1.5.3 Equal Circuits

Two useful gate equalities are[1]



(1.15)



(1.16)

2 Grover's Algorithm and its Limitations

2.1 Function

Grover's algorithm is known as a "search" algorithm or sometimes a "function inversion" algorithm. It can be applied to an unstructured search space made up of multiple unknown elements to search for one or more elements which fulfil a given property.

For example, Grover's algorithm can be used to search a system which is not necessarily in a classical state. For a quantum mechanical system made up of a superposition of states upon which an operator is applied, Grover's algorithm is able to search the component states even when the structure and identities of these elements are not already known before the measurement.

2.2 Process

The steps involved in running Grover's algorithm can be generalised as an initialisation of the basis states into a quantum system followed by two reflection transformations (producing a rotation) of this system. Over $\sqrt{2^n}$ iterations (where n is the total number of qubits) the algorithm works by rotating the system closer to a particular solution state. Through this process, the probability of the solution state is increased and those of other states reduced meaning that the system has a higher probability of resulting in the solution state upon measurement.

2.2.1 Initialisation of the Basis States

First, the quantum register must be constructed by applying the Hadamard gate to each of the qubits. For a system comprised of n qubits, there are a total of 2^n possible states. This is known as the state space. The quantum register $|s\rangle$ is then given by the uniform superposition over all states. The factor $\frac{1}{\sqrt{2^n}}$ means that all states have the same amplitude.

2.2.2 The Meaning of the Oracle

The first of the two transformations required to identify a solution is performed by a linear operator called the "Oracle", $\hat{O}$. The solution(s) to the search can be defined through a function which returns $f(|x'\rangle) = 1$ for an element $|x'\rangle$ which fulfils the search criteria, and $f(|x\rangle) = 0$ otherwise. This "checking" function is one part of the purpose of the Oracle as it identifies elements which will subsequently undergo a transformation. In

terms of the number of qubits, n, this stage can be represented as $|x\rangle, |x'\rangle \in \{0,1\}^n$. It is this part which gives Grover's algorithm the definition as a "function inversion" process as it uses a defined function in order to find elements which fulfil it.

When the oracle is applied to the superposition, $|s\rangle$, since it is a linear operator, it acts on each of the basis states, $|x\rangle$, individually and the summation is then over these results:

$$\hat{O}|s\rangle = \frac{1}{\sqrt{2^n}}\hat{O}\sum_{x=0}^{2^n-1}|x\rangle \tag{2.1}$$

The Oracle may be better understood by considering its effect on the basis states, $|x\rangle$, individually. For a single state, the operation can be represented as:

$$\hat{O}|x\rangle = (-1)^{f(x)}|x\rangle \tag{2.2}$$

For a non-solution $|x\rangle$, this state is unaffected by the Oracle:

$$\begin{aligned} f(x) &= 0; \\ \hat{O}|x\rangle &= (-1)^0|x\rangle \\ &= |x\rangle \end{aligned}$$

For a solution $|x'\rangle$, the operation returns:

$$\begin{aligned} \hat{O}|x'\rangle &= (-1)^1|x'\rangle \\ &= -|x'\rangle \end{aligned} \tag{2.3}$$

i.e, it negates the amplitude for $|x'\rangle$ and in doing so marks this state as a possible solution. Hence, applied to the superposition, the Oracle returns the sum over all $|x\rangle$ where $x \neq x'$, minus the state $|x'\rangle$:

$$\hat{O}|s\rangle = \frac{1}{\sqrt{2^n}}\left[\left(\sum_{x\neq x'}|x\rangle\right) - |x'\rangle\right] \tag{2.4}$$

The Born rule states that for a given solution, its probability density is given by the square of its amplitude[19]. This is the reason why, in initialisation, all states in the superposition have the same amplitude; since no information is known about any of the states, they must all have the same probability before measurement. This clearly remains the case even if one state has a negated amplitude and so a solution cannot be identified yet. Though the negated amplitude alone does not prove the means to identify the marked state, it nevertheless has the effect of reducing the average amplitude of the system.

2.2.3 Amplification and the Grover Diffusion Operator

In order for the marked state to be identified, it must have a high enough probability so that, upon measurement, the system collapses to this solution. The next stage of

Grover's algorithm increases the amplitude of the marked state (and thus its probability) while reducing the others. This transformation is made by the Diffusion operator given by $\hat{D} = 2\left|s\right\rangle\left\langle s\right| - 1$, which acts on the superposition following the Oracle. It increases the $-\left|x'\right\rangle$ amplitude by roughly three times and lowers the others accordingly so that the total probability still sums to 1. $\hat{D}$ is sometimes known as "inversion about the average" hence why it is significant that the average amplitude is lowered by the Oracle.

2.2.4 Solutions

Repeating the two transformation procedures, the system is rotated closer and closer to the solution. After $\sqrt{2^n}$ iterations, the system can be measured and the solution given in terms of its qubits.

2.3 Geometric Visualisation

The transformations performed by the Oracle and the Diffusion operator can be represented geometrically as a series of reflections in the 2D plane spanned by the system and the marked state.

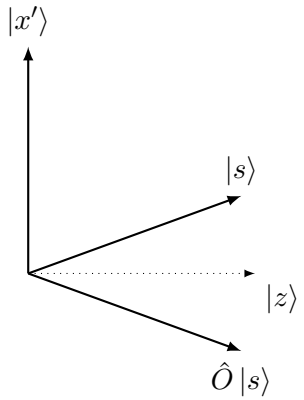


Figure 2.1: The Oracle operation represented as a reflection of the system $|s\rangle$ about an axis orthogonal to $|x'\rangle$

$|x'\rangle$ and $|s\rangle$ can be represented as a pair of vectors as shown above (fig. 1). If we consider a new horizontal axis $|z\rangle$ in the same plane, orthogonal to the vertical $|x'\rangle$, the Oracle acting on $|s\rangle$ can be considered as a reflection about $|z\rangle$. The following transformation by $\hat{D}$ reflects $\hat{O}\,|s\rangle$ about the original superposition vector (fig. 2).

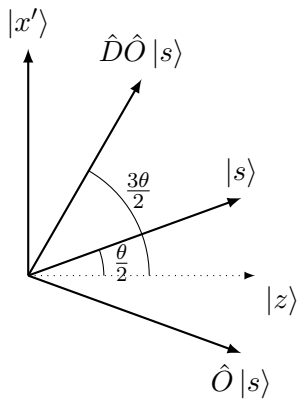


Figure 2.2: The reflection provided by the Diffusion operator on $\hat{O}\,|s\rangle$. $\frac{3\theta}{2}$ is the resultant angle of the transformed system.

2.4 Motivation for Grover's Algorithm

A significant motivation for the development of Grover's as a quantum algorithm is that it takes far fewer operations than a classical algorithm would require to solve the same problem. In fact, for a search which classically takes N operations, Grover's algorithm is able to perform it in $\sqrt{N}$ and hence the time required to complete the search is improved quadratically. This is because it is designed to check all component states simultaneously rather than one at a time as its classical counterpart would.

2.5 Applications

Grover's Algorithm has a number of very significant applications, giving it potential to be one of the most important quantum algorithms discovered.

One use is to solve a system of linear equations [20]. This involves a series of elementary row operations, performed by applying matrices obtained via Grover's Algorithm. Using Grover's compared to a classical algorithm speeds up the process substantially. It goes without saying that decreasing the time taken to solve a system of linear equations is extremely useful considering its wider implications as a solution to countless science and engineering tasks.

Some other applications of Grover's algorithm are; the solving of the collision problem[21], and estimating the mean and median of a set of numbers. It is also used in quantum cryptography[22]; it can be used to reverse-engineer cryptographic hash functions, which could potentially allow an attacker to find a victim's password or generate a set of counterfeit blocks.

2.6 Limitations

Despite the advantages Grover's algorithm can provide over classical mechanisms, it does not necessarily improve all classical processes and does sometimes not provide a practical improvement. This largely depends on the structure of the problem being solved. For example, the problem must feature a "black-box" function, such as $f(x)$ which was defined above. This is a function which can receive an input and produce a certain output but the way in which it operates is not necessarily known. Since Grover's works using indexing (since details of the database elements are unknown) it is required that the "black-box" is able to interpret the search elements correctly. Implementation of the Oracle can also pose a challenge as it must be compatible the particular hardware design of the processing machine. Better solutions where Grover's does not demonstrate a substantial improvement can be constructed using algorithms which take advantage of the structure of the particular search space[16] and can accommodate error factors such as noise[23] if these components are established and understood, completing the search in a shorter time and with more reliable solutions.

3 Quantum Error Correction

In the case of quantum computers, disruption of the states associated with bits is a significant issue. Small environmental disturbances, such as temperature fluctuations or dust, can cause a disruption of a qubit state. This is due to the small size of the physical qubits, such as atoms and spins, compared to the much larger size of transistors that make up a classical computer. The process of detecting and correcting errors on a quantum computer, in contrast to a classical computer, is further complicated by the fact that the qubits cannot be measured without collapsing their state. However, methods have been invented and constructed to overcome many of the issues, but not all. Most error-correcting codes that exist today are only able to handle a single error at a time. Therefore, in the following section, only single qubit errors are considered.

A basis for the understanding and construction of error-correcting codes is provided by the study of the evolution of qubits through their interaction with the environment. This will be studied in the next section. Some important ideas and tools are then presented in the following section, whereafter the favoured error-correcting code, the 7-qubit code[1], will be presented. Hereafter, the question of why the 7-qubit code is more favourable is discussed.

3.1 Interaction with the Environment

The environment around a qubit can be assigned a state, $|E\rangle$. When the environment interacts with the qubit, the result will be an entanglement of the form;

$$\begin{aligned} |E\rangle\,|0\rangle &\to \beta_0\,|E_0\rangle\,|0\rangle + \beta_1\,|E_1\rangle\,|1\rangle \\ |E\rangle\,|1\rangle &\to \beta_2\,|E_2\rangle\,|0\rangle + \beta_3\,|E_2\rangle\,|1\rangle \end{aligned} \tag{3.1}$$

where the amplitudes, $\{\beta_i\}_0^3$, determine the probabilities of the qubit remaining in the original state ($(\beta_0)^2$ or $(\beta_3)^2$), or evolving to the differing state ($(\beta_1)^2$ or $(\beta_2)^2$). However, qubits can be in a superposition of the two states, given by $\alpha_0\,|0\rangle + \alpha_1\,|1\rangle$. The entanglement with the environment then becomes

$$|E\rangle\,(\alpha_0\,|0\rangle + \alpha_1\,|1\rangle) \to \alpha_0\beta_0\,|E_0\rangle\,|0\rangle + \alpha_0\beta_1\,|E_1\rangle\,|1\rangle + \alpha_1\beta_2\,|E_2\rangle\,|0\rangle + \alpha_1\beta_3\,|E_3\rangle\,|1\rangle \tag{3.2}$$

[1] Also known as the *Steane* code, named after its inventor, Andrew Steane

Some algebra gives that Equation 3.2 can be written in the form

$$\begin{aligned}|E\rangle(\alpha_0|0\rangle+\alpha_1|1\rangle)\rightarrow&\frac{1}{2}(\beta_0|E_0\rangle+\beta_2|E_2\rangle)(\alpha_0|0\rangle+\alpha_1|1\rangle)\\&+\frac{1}{2}(\beta_0|E_0\rangle-\beta_2|E_2\rangle)(\alpha_0|0\rangle-\alpha_1|1\rangle)\\&+\frac{1}{2}(\beta_1|E_1\rangle+\beta_3|E_3\rangle)(\alpha_0|1\rangle+\alpha_1|0\rangle)\\&+\frac{1}{2}(\beta_1|E_1\rangle-\beta_3|E_3\rangle)(\alpha_0|1\rangle-\alpha_1|0\rangle)\end{aligned}$$

Further simplification yields

$$\begin{aligned}|E\rangle(\alpha_0|0\rangle+\alpha_1|1\rangle)\rightarrow&|A\rangle(\alpha_0|0\rangle+\alpha_1|1\rangle)\\&+|B\rangle(\alpha_0|0\rangle-\alpha_1|1\rangle)\\&+|C\rangle(\alpha_0|1\rangle+\alpha_1|0\rangle)\\&+|D\rangle(\alpha_0|1\rangle-\alpha_1|0\rangle)\end{aligned}$$

where $|A\rangle=\frac{1}{2}(\beta_0|E_0\rangle+\beta_2|E_2\rangle)$, $|B\rangle=\frac{1}{2}(\beta_0|E_0\rangle-\beta_2|E_2\rangle)$, $|C\rangle=\frac{1}{2}(\beta_1|E_1\rangle+\beta_3|E_3\rangle)$ and $|D\rangle=\frac{1}{2}(\beta_1|E_1\rangle-\beta_3|E_3\rangle)$ are states of the environment.

The interesting thing to note about the entanglement state is that each part corresponds to the transformation of the original qubit by the identity (**I**), a phase-flip (**Z**), a bit-flip (**X**), or a combination of the latter two (**Y**):

$$\begin{aligned}\alpha_0|0\rangle+\alpha_1|1\rangle&=\mathbf{I}(\alpha_0|0\rangle+\alpha_1|1\rangle)\\\alpha_0|0\rangle-\alpha_1|1\rangle&=\mathbf{Z}(\alpha_0|0\rangle+\alpha_1|1\rangle)\\\alpha_0|1\rangle+\alpha_1|0\rangle&=\mathbf{X}(\alpha_0|0\rangle+\alpha_1|1\rangle)\\\alpha_0|1\rangle-\alpha_1|0\rangle&=\mathbf{Y}(\alpha_0|0\rangle+\alpha_1|1\rangle)\end{aligned}$$

Therefore, a qubit under interaction with the environment can stay unaffected or be affected by either a phase-flip, bit-flip or both.

3.2 Codewords and Ancillary Qubits

The essence of quantum error detection lies in the use of codewords and ancillary qubits. Codewords, noted by $|\bar{x}\rangle$ for $x=0,1$, are a collection of qubit states encoded from ground state qubits. The encoding circuit has different form depending on the desired error-correcting code. After completion of the encoding circuit, the gate(s), for which one expects an error to possibly occur, is (are) applied to all the codewords. The error detection circuit can then be executed and hereafter the error can be corrected classically, or in some cases, as it is with the 3-qubit code, by automatic application of an error-correcting circuit.

The simplest encoding circuit for the codewords is applicable for the 3-qubit code. The encoding circuit functions as a "copying" circuit. The '*No Cloning*'-theorem prevents

from directly cloning a qubit, but by applying CNOT gates to ground state qubits, with the original qubit as the control qubit, one can get a collection of simple $|0\rangle$- or $|1\rangle$-qubits. The general formula for this encoding circuit, when one desires n total qubits (including the original qubit) in the $|x\rangle$ state, is given by

$$|\bar{x}\rangle_n = \mathbf{C}_{(n-1)n(n-2)}...\mathbf{C}_{n0}\,|x\rangle\,|0\rangle_{n-1}\,...\,|0\rangle_0 = |x_{n-1}...x_0\rangle \tag{3.3}$$

For the 3-qubit state $n = 3$ and just 2 CNOT gates are required. For the 7-qubit code, the encoding circuit is not so simple, but this will be explored in the section for the 7-qubit code.

For a classical computer the story can end here; one measures all codewords and the state for which the majority of the codewords are found to be in, is, under the right assumptions and circumstances, more likely to be uncorrupted state. This can also be extended to longer codewords to allow for correcting several error-occurrences at once. Of course this is not the solution for the case of the quantum computer.

Instead of performing direct measurements on qubits, ancillary qubits are created, from ground state qubits, and used in a circuit, specific to the different error-correcting codes, along with the codewords to provide information about the relations between the codewords - similar to the classical situation where one looks at the codewords and determines the majority-state.

3.3 The 7-qubit Error-Correcting Code

The 7-qubit code consists of 6 ancillas and 6 unitary, commuting operators. The operators are as follows

$$\mathbf{M}_0 = \mathbf{X}_0\mathbf{X}_4\mathbf{X}_5\mathbf{X}_6 \quad \mathbf{M}_1 = \mathbf{X}_1\mathbf{X}_3\mathbf{X}_5\mathbf{X}_6 \quad \mathbf{M}_2 = \mathbf{X}_2\mathbf{X}_3\mathbf{X}_4\mathbf{X}_6$$

$$\mathbf{N}_0 = \mathbf{Z}_0\mathbf{Z}_4\mathbf{Z}_5\mathbf{Z}_6 \qquad \mathbf{N}_1 = \mathbf{Z}_1\mathbf{Z}_3\mathbf{Z}_5\mathbf{Z}_6 \qquad \mathbf{M}_2 = \mathbf{Z}_2\mathbf{Z}_3\mathbf{Z}_4\mathbf{Z}_6$$

The $\mathbf{M}$-operators are used in the encoding circuit to get the codewords for the 7-qubit code. The codewords are given by[4]

$$|\bar{0}\rangle_{cw} = \frac{1}{\sqrt{8}}(\mathbf{I}+\mathbf{M}_0)(\mathbf{I}+\mathbf{M}_1)(\mathbf{I}+\mathbf{M}_2)\,|\bar{0}\rangle_7$$

$$|\bar{1}\rangle_{cw} = \frac{1}{\sqrt{8}}(\mathbf{I}+\mathbf{M}_0)(\mathbf{I}+\mathbf{M}_1)(\mathbf{I}+\mathbf{M}_2)\,|\bar{1}\rangle_7 \tag{3.4}$$

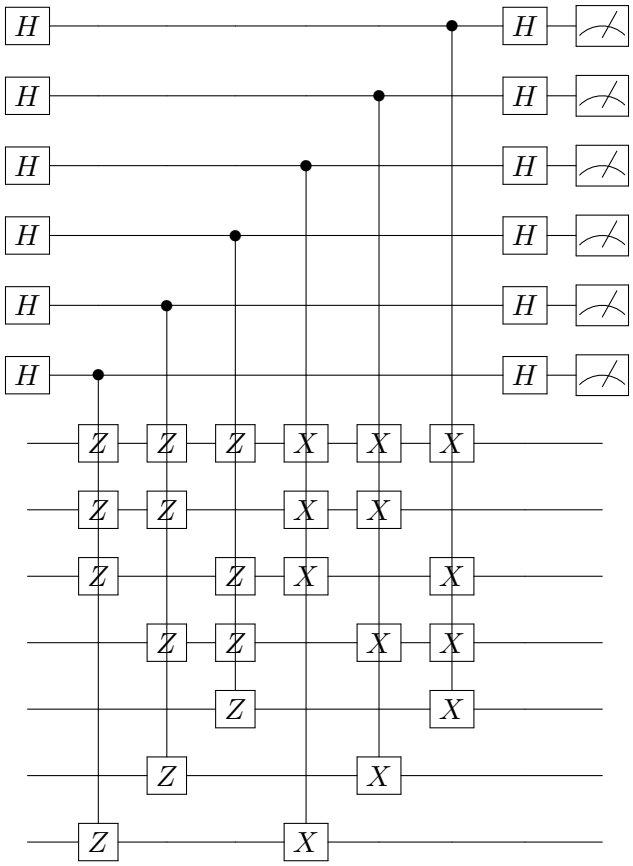
where $|\bar{0}\rangle_7, |\bar{1}\rangle_7$ are as defined by Equation 3.3. Some lengthy algebra shows that the codewords are

$$\begin{aligned}|\bar{0}\rangle_{cw} = \frac{1}{\sqrt{8}}(\,&|0000000\rangle + |1010101\rangle + |0110011\rangle + |1100110\rangle + \\ &|0001111\rangle + |1011010\rangle + |0111100\rangle + |1101001\rangle\,)\end{aligned} \tag{3.5}$$

$$|\bar{1}\rangle_{cw} = \frac{1}{\sqrt{8}}(\,|1111111\rangle + |0101010\rangle + |1001100\rangle + |0011001\rangle + |1110000\rangle + |0100101\rangle + |1000011\rangle + |0010110\rangle\,) \tag{3.6}$$

The 6 ancillas are entangled with these codewords by the circuit shown in Equation 3.8. The application of the Hadamard gates is nescessary to entangle the ancillas properly with the codewords, as illustrated in Equations 1.15, 1.16. Each operation (between the Hadamard gates) is easily seen to be equivalent to one of the 6 operators with a control qubit, consisting of one of the ancillas:

$$\mathbf{c}_5\mathbf{M}_2\mathbf{c}_4\mathbf{M}_1\mathbf{c}_3\mathbf{M}_0\mathbf{c}_2\mathbf{N}_2\mathbf{c}_1\mathbf{N}_1\mathbf{c}_0\mathbf{N}_0\,|\bar{x}\rangle_{cw} \tag{3.7}$$



(3.8)

After completing the circuit of Equation 3.8, the measurement results of the ancillas will reveal the relations between the codewords. The results are known as *the error syndrome*. The possible outcomes, due to a bit-flip, phase-flip or the combination on a single qubit, are listed in Table 3.1, where the subscript indicates which qubit the error has occurred on. Once the error syndrome has been analyzed, the error can be corrected by applying the appropriate **X**-, **Z**- or **Y**-gate to revert the error. E.g. if the output of the measurements of the ancillas is $|000100\rangle$, then an **X**-gate should be applied to the 4th codeword qubit (i.e. $\mathbf{X}_3$).

Syndrome	Error	Syndrome	Error	Syndrome	Error
$\|000001\rangle$	X_0	$\|001000\rangle$	Z_0	$\|001001\rangle$	Y_0
$\|000010\rangle$	X_1	$\|010000\rangle$	Z_1	$\|010010\rangle$	Y_1
$\|000011\rangle$	X_2	$\|011000\rangle$	Z_2	$\|011011\rangle$	Y_2
$\|000100\rangle$	X_3	$\|100000\rangle$	Z_3	$\|100100\rangle$	Y_3
$\|000101\rangle$	X_4	$\|101000\rangle$	Z_4	$\|101101\rangle$	Y_4
$\|000110\rangle$	X_5	$\|110000\rangle$	Z_5	$\|110110\rangle$	Y_5
$\|000111\rangle$	X_6	$\|111000\rangle$	Z_6	$\|011111\rangle$	Y_6

Table 3.1: Error Syndrome for the 7-qubit Code[24]

3.4 Advantages and Disadvantages to Error-correcting Codes

Throughout this chapter, the 3-qubit code has been mentioned several times. This code is of course a lot shorter and simpler than the presented 7-qubit code, so one might wonder why not the 3-qubit code is favoured over the 7-qubit code. The answer is simple and just requires to consider a basic constraint for any (single qubit error) error-correcting code. The analysis of the evolution of a qubit in interaction with its environment revealed the possibility of 3 kinds of corruptions. This means that for n qubits there is 1 uncorrupted state and $3n$ possible corrupted states. The dimension of the subspaces associated with these possible states is $2(1 + 3n)$, and the dimension of the space associated with the states of n qubits is 2^n. The dimension of the subspaces must not be higher than that of all possible states, and so this puts a constraint on the minimum number of qubits required to correct any one of the 3 possible errors:

$$2^n \geq 2(1 + 3n) \tag{3.9}$$

This is not true for $n = 3$. However, if it can be assumed that only one kind of error can occur, the constraint becomes

$$2^n \geq 2(1 + n) \tag{3.10}$$

This is true for $n = 3$, and therefore a 3-qubit code can be used if it can be assumed that only one kind of error can occur. In reality, this is not very useful, as it's very hard or often impossible to control the environment in such a way.

The lowest number that satisfies Equation 3.9 is not 7, but 5, for which it holds as equality. Indeed a 5-qubit error-correcting code exists and can be used to detect and correct an error of any of the three types. However, as it turns out, a lot of logical gates are more easily and transparently applied to the 7-qubit code than the 5-qubit, and this is primarily why the 7-qubit code is more often the favoured one.[4][1]

Any code larger than the 7-qubit code is generally an unnecessary complication[2]. Also since errors can also occur during the error-detecting circuits, it's valuable to keep the

[2]The 9-qubit code, also known as Shor's code, named after its inventor, was in fact the first invented full quantum error-correcting code, but has no significant advantage over the later invented 7-qubit code. Today, the 9-qubit code is primarily mentioned for the historical significance[1]

risk of this down by keeping the number of susceptible qubits down.

4 Shor's Algorithm

4.1 Motivation and speed-up over classical algorithms

Shor's Algorithm is a quantum algorithm designed for integer factorisation. It was formulated in 1994 and named after mathematician Peter Shor. The exact problem we are trying to solve via Shor's algorithm is this: when given a composite number N, find an integer d that divides by N and is between 1 and N. 3

RSA cryptosystem[25] is used in secure data transmission, it is effective because it relies on the fact that factorising extremely large numbers is "hard". As an application of Shor's algorithm means that N can be factorised in polynomial time, this would effectively break RSA.

4.2 Procedure

The algorithm comprises a classical part and a quantum part:

4.2.1 Classical part

Shor's algorithm reduces the problem of finding the prime factors of a large number N to that of finding the period of the function $f(x) = a^x \mod N$ with a and x integers. The algorithm splits into two parts: a main classical loop, and a quantum period-finding function which takes advantage of the exponential speed-up afforded by the quantum Fourier transform. The classical part of the function can be divided into the following simple steps:

1. Pick a random number $a < N$.
2. Calculate $\gcd(a, N)$ using the efficient (classical) 'Euclidean algorithm'. If this is equal to one, then this already gives a nontrivial factor of N, and we are finished.
3. Otherwise, if $\gcd(a, N) \neq 1$ then we use the quantum function described in the following section to find the period r, of the function $f(x) = a^x \mod N$.
4. If this value of r turns out to be even, then $\gcd(a^{r/2} + 1, N)$ and $(a^{r/2} - 1, N)$ are both nontrivial factors, and we are done. Otherwise, repeat the function for another random value of a.

Given more time, we would have implemented this function in our code along similar lines to the '`sympy`' version of Shor's algorithm although with a different design structure focusing on objects acting on the register state.

4.2.2 The quantum Fourier transform

The discrete Fourier transform of a function $f : \mathbb{Z}^N \to \mathbb{C}$ is

$$\widehat{f}(k) = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} f(x)\omega^{kx} \quad \text{with} \quad \omega = \exp\left(-\frac{2\pi i}{N}\right), \tag{4.1}$$

$$\text{with inverse:} \quad f(x) = \sqrt{N} \sum_{k=0}^{N-1} \widehat{f}(k)\omega^{-kx}. \tag{4.2}$$

Where $N = 2^n$, this defines a gate $\mathcal{F}_N$ which acts on n qubits and which can be represented in the computational basis by an $N \times N$ matrix:

$$\mathcal{F}_N : \begin{bmatrix} f_0 \\ f_1 \\ \vdots \\ f_N \end{bmatrix} \mapsto \begin{bmatrix} \widehat{f_0} \\ \widehat{f_1} \\ \vdots \\ \widehat{f_N} \end{bmatrix} \quad \text{with} \quad (\mathcal{F}_N)_{ab} = \frac{\omega^{ab}}{\sqrt{N}} \quad \text{and} \quad \mathcal{F}_N = \frac{1}{\sqrt{N}} \begin{bmatrix} 1 & 1 & \dots & 1 \\ 1 & \omega & \dots & \omega^{N-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{N-1} & \dots & \omega^{(N-1)^2} \end{bmatrix} \tag{4.3}$$

Here $\mathcal{F}$ is clearly unitary, since $\left[\mathcal{F}_N^\dagger \mathcal{F}\right]_{ab} = \frac{1}{N} \sum_{x=0}^{N-1} \omega^{(a-b)x} = \delta_{ab}$. Naively, it would seem that acting on a state vector in $\mathbb{C}^{2n}$ with the (inverse) Fourier transformation matrix should require $\mathcal{O}(2^{2n})$ calculations. However, the (classical) 'fast Fourier transform' only requires $\mathcal{O}(n2^n)$ calculations, and this can be reduced further still by working with qubits instead of classical bits. This is because the quantum fourier transform can be decomposed into the tensor product of a series of phase-shift gates R_ϕ acting on individual qubits in equal superposition as follows:

$$\begin{aligned}
\mathcal{F}_{(2^n)}^{-1} |x\rangle &= \frac{1}{2^{\frac{n}{2}}} \sum_{k=0}^{2^n-1} \exp\left(\frac{2\pi i k x}{2^n}\right) |k\rangle \qquad \text{Since} \quad k = \sum_{j=0}^{n-1} 2^j k_j \,, \\
&= 2^{-\frac{n}{2}} \sum_{(k_0,\dots k_{n-1})\in\{0,1\}^n} \left[\prod_{j=0}^{n-1} \exp\left(\frac{2\pi i x\, k_j}{2^{n-j}}\right) |k_{n-1}\rangle \otimes \cdots \otimes |k_1\rangle \otimes |k_0\rangle\right] \\
&= 2^{-\frac{n}{2}} \bigotimes_{j=0}^{n-1} \sum_{k_j=0,1} \exp\left(\frac{2\pi i x k_j}{2^{n-j}}\right) |k_j\rangle \quad \text{(tensor product taken with } j \textit{ decreasing} \text{ left to right)} \\
&= \bigotimes_{j=0}^{n-1} \frac{1}{\sqrt{2}} \left(|0\rangle + \exp\left(\frac{2\pi i x}{2^{n-j}}\right) |1\rangle\right) \\
&= \bigotimes_{j=0}^{n-1} R_{(x\theta_j)} H_d |0\rangle \quad \text{defining} \quad \theta_j = \frac{2\pi}{2^{n-j}}\,.
\end{aligned} \tag{4.4}$$

This means that the quantum implementation of the Fourier transform is exponentially more efficient than the classical version, as it can be implemented using just $\mathcal{O}(n^2)$ quantum gates.

4.2.3 The quantum period finding function

Note: this section closely follows the description of [26] with a few changes in notation, and is included below for reference only. The inputs to the quantum period-finding function are the large number, N, that we wish to factorise, together with an arbitrary integer $a < N$. The algorithm proceeds in the following steps:

1. Given a and N, define $q = \lceil \log_2 N \rceil$ as the number of bits required to encode N, and $n = 2q$. Now define a *pair* of quantum registers Ψ_n and Ψ_q of lengths n and q respectively, with the first initialised in the equal-superposition state using Hadamard gates, and the second in zero state:

$$\Psi_n \otimes \Psi_q = (H_d^{\otimes n} \otimes I^{\otimes q})(|0\rangle^{\otimes n} \otimes |0\rangle^{\otimes q}) = \frac{1}{2^{n/2}} \sum_{x=0}^{2^n-1} |x\rangle \otimes |0\rangle^{\otimes q} \tag{4.5}$$

   Let $f : \{0, 1, \ldots, 2^n\} \to \{0, 1, 2^q\}$ be a function mapping the 2^n basis states of Ψ_n to the 2^q of Ψ_q with $f(x) = a^x \mod N$.

2. Define the computational gate U_f, which acts on states $|x\rangle \in \Psi_n$, $|y\rangle \in \Psi_q$ as:

$$U_f : |x\rangle \otimes |y\rangle \mapsto |x\rangle \otimes |y + f(x)\rangle \ . \tag{4.6}$$

   In principle it is always possible to construct a quantum gate acting on n qubits to arbitrary precision out of a universal set of gates, and according to [26] this can be done using just $\mathcal{O}(n^3 \log n \log\log n)$ elementary gates. In practice, this is not trivial, but assuming such a U_f can be constructed, then its action on the pair of registers is:

$$U_f(\Psi_n \otimes \Psi_q) = \frac{1}{2^{n/2}} \sum_{x=0}^{2^n-1} |x\rangle \otimes |f(x)\rangle \qquad |x\rangle \in \Psi_n \quad , |f(x)\rangle \in \Psi_q \, . \tag{4.7}$$

3. Perform a measurement on the second register, projecting it into a basis state $|s\rangle$ with $s \in \{0, 1, \ldots, 2^q\}$. The probability of collapse into a given state is given by

$$P(|s\rangle) = \frac{|f^{-1}(s)|}{2^{n/2}} \quad \text{where} \quad |f^{-1}(s)| \neq 0 \, , \tag{4.8}$$

   and after measurement the system is in the state

$$\left(\sum_{x \in f^{-1}(s)} |x\rangle \right) \otimes |s\rangle = \left(\sum_{x=0}^{2^n-1} \chi_s(x) \, |x\rangle \right) \otimes |s\rangle \qquad \text{with} \quad \chi_s(x) = \frac{1}{|f^{-1}(s)|} \, . \tag{4.9}$$

4. Apply the quantum Fourier transform to the first register by acting with $\mathcal{F}_{2^n} \otimes I^{\otimes n}$, putting the system into a state

$$\sum_{k=0}^{2^n-1} \widehat{\chi_s}(k) \, |k\rangle \otimes |s\rangle \tag{4.10}$$

   As mentioned in [ref], this can be achieved in just $\mathcal{O}(n^2)$ gates by decomposing the QFT into a series of single-qubit gates.

5. Measure the first register, Ψ_n, which will be in state $|k\rangle$ with probability $|\widehat{\chi_s(k)}|^2$. From this measured state, c, it is shown in [26] that the period r of f can be found using the method of continued fractions.

In order to implement Shor's Algorithm efficiently, we would first need to change the state-measurement method so that it acts as a (non-unitary) gate projecting the state vector onto the complex line. This would allow us to extend the measurement method to one that only measured the first k qubits of the system, so that we could easily work with a double register. The second, larger, challenge is working out how to construct the gate $U_f : |x\rangle |y\rangle \mapsto |x\rangle |y \oplus f(x)\rangle$ in terms of fundamental gates. In principle, it is possible to construct U_f from fundamental gates using the Solovay-Kitaev algorithm, but in practice we could not work out how to do this (in the `sympy` documentation, this is implemented as the black-box function `CMod`).

5 Program Structure and Design

5.1 Object Oriented Programming

The object oriented programming language Python is the best tool to build an application like this because it favours modular design, in a group dynamic everyone can contribute to something that can be easily connected together increasing productivity.

5.2 Data structures

5.2.1 Qubit

The qubit is the simplest of data structures created, at instantiation each qubit is assigned to the default $|0\rangle$ state, on its own a single qubit isn't too useful, it is possible to act on single qubit gates in this form, but this limits us to very basic systems. Its also possible in this form to measure a single qubit's state, this causes it to collapse into either $|0\rangle$ *or* $|1\rangle$ depending on the probabilities manipulated by a system of single qubit gates.

5.2.2 Register

The Register object stores qubits state data, from which it is possible to apply comparatively more operations than a single qubit, a register is a collection of n qubits stored as their tensor product. By storing the state of all the qubits in this tensor it enables us to use multiqubit gates. It stores operations that are going to act upon it, as a two dimensional stack like array, where each element represents a set of gates in order of execution. The register also computes all the operations acting on it in the stack by calling the .applyGates() method and has the ability to change the initial individual qubit states from the default $|0\rangle$ state.

5.2.3 Tensor Product

The tensor product of a pair of two-dimensional matrices A and B, of dimensions (a_m, a_n) and (b_m, b_n) respectively, is an $(a_m \cdot b_m, a_n \cdot b_n)$-dimensional matrix $A \otimes B$, with elements given as

$$A \otimes B = \begin{bmatrix} A_{00}[B] & A_{01}[B] & \dots & A_{0a_n}[B] \\ A_{10}[B] & A_{11}[B] & \dots & A_{1a_n}[B] \\ \vdots & \vdots & \ddots & \vdots \\ A_{a_m0}[B] & A_{a_m1}[B] & \dots & A_{a_ma_n}[B] \end{bmatrix} = \begin{bmatrix} A_{00}B_{00} & A_{00}B_{01} & \dots & A_{0a_n}B_{0b_n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{a_m0}B_{b_m0} & A_{a_m0}B_{b_m1} & \dots & A_{a_ma_n}B_{b_mb_n} \end{bmatrix} \tag{5.1}$$

In the code, this is used to take the tensor product of both qubits in the register and gates, and is implemented as follows (Note the order of operation is essential as it determines the convention for the state of multiple qubits) :

```
def SingleOperatorTensor(A, B):
    """Return the tensor product of two operators
    A and B in the computational basis."""
    A = np.array(A) # in case items passed as lists
    B = np.array(B)
    if len(np.shape(B))==1: # convert to 2-d matrices
        B = B.reshape(len(B),1)
    if len(np.shape(A))==1:
        A = A.reshape(len(A),1)
    B_tmp = np.tile(B, np.shape(A))
    A_tmp = A.repeat(np.shape(B)[0], axis=0).repeat(np.shape(B)[1], axis=1)
    return np.multiply(A_tmp, B_tmp)

def OperatorTensor(mylist):
    """Apply the tensor product of qubits
    from an input array of operation gates"""
    result = [[1]] # Any matrix M tensored with [[1]] is just M
    for q in reversed(mylist):
        result = SingleOperatorTensor(q, result)
    return result
```

So that, for instance, a register of length three in the 'zero' state could be set to the state `OperatorTensor([zero]*3)` where `zero` has been defined as `zero = np.array([[1],[0]])`. This encodes the operation:

$$\begin{bmatrix}1\\0\end{bmatrix} \otimes \begin{bmatrix}1\\0\end{bmatrix} \otimes \begin{bmatrix}1\\0\end{bmatrix} = \begin{bmatrix}1 & 0 & 0 & 0 & 0 & 0 & 0 & 0\end{bmatrix}^T . \tag{5.2}$$

This generalises to arbitrary matrix operators. For example, the following code would act on a three-qubit register state with the operator $H_d \otimes X \otimes H_d$:

```
hd = Gates.Hadamard()
gate_X = Gates.X()
reg = Register(3) # create a 3-qubit register
reg.addGates([hd,gate_X,hd]) # take the tensor product of the three gates
reg.applyGates()
```

Here, the `addGates` method returns the tensor product of operators as defined above, while `applyGates` multiplies the state of the register by all matrices in the list: `reg.operations`.

```
    def addGates(self, list_of_gates):
        # Adds a row of gates to the list of operations
        gates = []
        for x in range(len(list_of_gates)):
            print(list_of_gates[x].name)
            gates.append(list_of_gates[x].gate)
        operator = Qbit.OperatorTensor(gates)
```

```
assert np.shape(operator)==(self.statedim, self.statedim),
    "operator is wrong dimension"
self.operations.append(operator)
```

5.2.4 Gates

Abstract Methods

All gates are defined as being unitary, creating an abstract object class sets the standard for all gates ensuring the behaviour exhibited is expected and uniform. This also serves to allow for object inheritance within the code leading to higher efficiency. All gates are assigned an identifying string, a number of qubits to act upon and a gate matrix to act on a tensored qubit register.

Standard Gates

The following table summarises the gates used in our program and how they are represented in the circuit diagram notation, the multiqubit gates have been left out due to the size of their respective matrix representation. Here the phase shift gate is the only operator which takes in an argument. The gates operate on the tensored qubit register state through a matrix multiplication, but in order for the dimensions to be correct the gates themselves have to be tensored together, this is done using the same method as the qubit, it is also important to note that to designate the operator to a specific qubit the order in which the operators are tensored is imperative, the first element in the list of gates will act on the first qubit in the register.

Name	Python object	Notation in computational basis	Circuit diagram representation
Identity	`Gates.Id()`	$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$	——
Hadamard	`Gates.Hadamard()`	$H_d = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$	$\boxed{H_d}$
NOT Gate/ X	`Gates.X()`	$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$	$\boxed{X}$
Y	`Gates.Y()`	$Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$	$\boxed{Y}$
Spin-flip/ Z	`Gates.Z()`	$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$	$\boxed{Z}$
Phase Shift	`Gates.PhaseShift(phi)`	$R_\phi = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\phi} \end{bmatrix}$	$\boxed{R_\phi}$
S	`Gates.S()`	$S = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$	$\boxed{S}$
$S^\dagger$	`Gates.SHerm()`	$S^\dagger = \begin{bmatrix} 1 & 0 \\ 0 & -i \end{bmatrix}$	$\boxed{S^\dagger}$
T	`Gates.T()`	$T = \begin{bmatrix} 1 & 0 \\ 0 & \frac{1+i}{\sqrt{2}} \end{bmatrix}$	$\boxed{T}$
$T^\dagger$	`Gates.THerm()`	$T^\dagger = \begin{bmatrix} 1 & 0 \\ 0 & \frac{1-i}{\sqrt{2}} \end{bmatrix}$	$\boxed{T^\dagger}$

5.3 Quantum Circuit and Lane Implementation

The register handles the operations the same way as reading a circuit lane diagram, it maps out the gates to their corresponding qubits to act upon, with each column representing a single time-step, implemented by a stack procedure to execute each column, or set, successively.

$$\begin{array}{ll} |\psi_1\rangle & \text{—}\boxed{H}\text{———}\boxed{H}\text{———} \\ |\psi_2\rangle & \text{—}\boxed{H}\text{———}\boxed{H}\text{———} \\ |\psi_3\rangle & \text{—}\boxed{H}\text{———}\boxed{H}\text{———} \end{array} \tag{5.3}$$

This diagram shows the implementation of two sets of the standard Hadamard Gates acting on a register made up of three qubits. Reading left to right follows the convention of the order of execution. The gates are tensored together to make a $8x8$ matrix which acts on the register state by matrix multiplication: $H^{\otimes 3} \cdot \psi^{\otimes 3}$

5.4 Grover's

Object Class

It was decided that the best approach to implement Grovers was to create an object that takes in a register and creates the quantum circuit specifically for it's input register. The Grover class has three main sets of gates, the first being the initial Hadamards, this arranges all the qubits in the register into equal superposition as standard. The next two specialised sets are repeated for a user specified number of iterations, this according to the theory should ideally be $\sqrt{N}$, otherwise the algorithm will over shoot computing a worse probability.

Grover's Gates

The two special operators that are repeated necessary to implement Grover's Algorithm are:

- Oracle, $O\left|x\right\rangle = (\text{-}1)^{f(x)}\left|x\right\rangle$.
  This identifies the target location and negates it's amplitude, it's matrix representation is similar to the identity matrix but the element that corresponds to the target item is negative.
- Grover Diffusion, $D = 2\left|s\right\rangle\left\langle s\right| - 1$ where $\left|s\right\rangle = H^{\otimes N}\left|0\right\rangle^{N}$
  This is represented by a full matrix where every element is $2/N$ minus the identity matrix. This amplitude amplification stage is corresponds to a geometric reflection, together with the oracle operator applies a rotation.

Additional Grover Functionality

To bring all this together the Grover class has certain functions that handle the data, this includes the following:

- `collectOperators`, this function loops over the creation functions to provide the correct set of gates as requested by the user in a format ready to be added to the register operation stack.
- `run`, this function takes in the register that the operators are going to act on, collects the number of gates depending on how many iterations have been chosen and updates the register accordingly.

5.5 Quantum Error-Correcting Code Implementation

Given the way that the code is constructed, an implementation of a quantum error-correcting code will require the ability for two separate registers to interact and get entangled (a codeword register and an ancillary register, the latter to be measured). This is not possible for the code as it is constructed now. The code is only able to interact in

between a single register. This means that to implement the 7-qubit code, a 13-qubit register must be created, and $2^{13} \times 2^{13}$ matrices must be applied to it. This would require a vast amount of time to set up, construct and run.

However, the biggest issue is with the measurement. The register is not build to "separate" into single qubits and measure individual qubits, but only to measure the entire register. This means that one would measure all qubits, including the codeword qubits, and by doing this the concept of the simulator looses its meaning - it would no longer act as a quantum computer, but as a classical computer for which the user can "pick out" whatever they want to know.

Given more time, it would have been interesting to extent the code, such that a register can interact with another register. This would mean that the gates as seen in Equation 3.8 could be applied as intended without the need for a $2^{13} \times 2^{13}$ matrix. It would also solve the measurement issue, as the allowed interaction between registers would give the opportunity to measure certain registers and leave others unmeasured.

5.6 Interfacing and Assertion Statements

For testing and debugging the code, assertion statements provide protection against errors, such as if for example incorrect data types are being passed through as arguments or wrong dimensions for matrix algebra, python will raise an exception error telling the user what condition has not been met. This saves time from following data through various functions allowing modules to be coupled together with ease. The assertion statements can be switched off after the final version of the code has been produced to improve performance. An example of this would be checking the tensor of a set of gates creates a matrix of correct dimensions for a matrix multiplication on the register's state.

```
def addGates(self, list_of_gates):
    #...
    assert np.shape(operator)==(self.statedim, self.statedim), "operator is wrong dimension"
    #...
```

6 Groupwork Comments

Programming the simulator for a circuit-model quantum computer and accompanying elements posed a unique challenge. Not only was the subject matter and implementation difficult, we also had to effectively work as a group. Initially the group harboured ideas of all contributing the exact same amount to each section. It quickly became apparent that this resulted in too wide a spread for each of member and what is colloquially referred to as *too many cooks in the kitchen*. Realising the futility of continuing in this manner; individuals were instead assigned to areas where they were strongest and given smaller portions of the other sections. This proved to be a superior method of teamwork as individuals were able to focus more wholly on their sections as well as not interrupting the work of others.

6.1 Tools used

Group programming poses unique challenges regarding cooperation, understanding, interpretation, and writing the code itself. Collaborative coding is difficult due to how code is always a hierarchy of pieces that depend on each other, thus small changes by one programmer may unwittingly affect the work done by others or have unseen ramifications in other parts of the code. As such, it is important to lay down standard practices and clearly communicate the division of tasks. A tool that was vastly useful for these tasks was git, particularly in conjunction with gitlab. Gitlab was used as an online hub for repository management and also served as an aid in maintaining the aforementioned organisation and standardisation. Another important aspect of group programming and large project in general is version control. Using robust git practices in conjunction with gitlab and effective communication an robust and effective history was maintained with integrity.

6.2 Programming style

The earlier section on Design Choices details some of the decisions and reasoning surrounding the the 'shape and flow' of the code. The programming section was a particular area where it became apparent that separation was required to effectively work, that is to say, effective division of tasks was essential so as to not have overlapping or conflicting code. One issue that came up when dividing the tasks for the programming was the lack of knowledge as to how difficult each section would be. This was countered by starting with theory work so individuals could have an understanding of each section and then discussing potential implementation methods to ascertain how difficult we thought each

section would be. A core principle that was followed in the design was that of minimum viable product (MVP). Before adding excess complexity at each stage, it was important to ensure the code was capable of the bare minimum required. MVP is an effective method to guarantee that even if something is attempted incorrectly, there remains the ability to revert back to the basic functionality. Another design method we adhered to involved completion of basic hand drawn flow charts before beginning to code. This allows for an overall picture of what the code will do and more importantly how it will do it before the first line is written. This has the effect of allowing for modular and efficient design by providing an overview at all times. An important tenet of the design method used in this report was Unit Testing. Unit testing revolves around testing individual smaller modules of the source code at each time. It can be thought of as checking each cog in a machine functions properly. Unit testing has two-fold benefit not only could individual parts be ensured to work but also if a larger test of functionality resulted in an error unit tests could be run in conjunction with conventional debugging to determine where functionality broke down. Unit testing served as an excellent method for maintaining code integrity and ensuring functionality especially as the code grew in complexity. (The exported code does not include the unit tests as they are not designed to be user facing but rather for development)

Bibliography

[1] Michael A. Nielsen and Isaac L. Chuang. *Quantum Compuatation and Quantum Information*. Cambridge University Press, 2010.

[2] Jane Weir. *Max Planck: Revolutionary Physicist*. Capstone, 2009.

[3] Albert Einstein. "On a heuristic point of view about the creation and conversion of light". In: *Annalen der Physik* 17.6 (1905), pp. 132–148.

[4] N. David Mermin. *Quantum Computer Science*. Cambridge University Press, 2007.

[5] Stephen Wiesner. "Conjugate coding". In: *ACM Sigact News* 15.1 (1983), pp. 78–88.

[6] Rafael D Sorkin. "Quantum mechanics as quantum measure theory". In: *Modern Physics Letters A* 9.33 (1994), pp. 3119–3127.

[7] Paul Benioff. "The computer as a physical system: A microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines". In: *Journal of statistical physics* 22.5 (1980), pp. 563–591.

[8] David Deutsch. "Quantum theory, the Church–Turing principle and the universal quantum computer". In: *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences* 400.1818 (1985), pp. 97–117.

[9] Lov K Grover. "A fast quantum mechanical algorithm for database search". In: *arXiv preprint quant-ph/9605043* (1996).

[10] Richard Tanburn, Emile Okada, and Nike Dattani. "Reducing multi-qubit interactions in adiabatic quantum computation without adding auxiliary qubits. Part 1: The" deduc-reduc" method and its application to quantum factorization of numbers". In: *arXiv preprint arXiv:1508.04816* (2015).

[11] Zeeya Merali. "Spooky steps to a quantum network". In: (2006).

[12] Armine Hareyan. "Qubits poised to reveal our secrets". In: (2007).

[13] John Markoff. "Physicists Create a Working Transistor From a Single Atom". In: *The New* (2012).

[14] E Connover. "Google Moves toward Quantum Supremacy with 72-qubit Computer". In: *Science News* 193.6 (2018), p. 13.

[15] Jacob Aron. "IBM unveils its first commercial quantum computer". In: *Newscientist* (2019).

[16] George F Viamontes, Igor L Markov, and John P Hayes. "Is quantum search practical?" In: *Computing in science & engineering* 7.3 (2005), pp. 62–70.

[17] P Kumar, S Sendelbach, MA Beck, JW Freeland, Zhe Wang, Hui Wang, C Yu Clare, RQ Wu, DP Pappas, and R McDermott. "Origin and reduction of 1/f magnetic flux noise in superconducting devices". In: *Physical Review Applied* 6.4 (2016), p. 041001.

[18] Christopher M Dawson and Michael A Nielsen. "The solovay-kitaev algorithm". In: *arXiv preprint quant-ph/0505030* (2005).

[19] Max Born. "Zur Quantenmechanik der Stovorgnge". In: *Zeitschrift fr Physik* 37.12 (Dec. 1926), pp. 863–867. DOI: 10.1007/bf01397477. URL: http://dx.doi.org/10.1007/BF01397477.

[20] Karthik Srinivasan, Bikash Behera, and Prasanta Panigrahi. "Solving Linear Systems of Equations by Gaussian Elimination Method Using Grover's Search Algorithm: An IBM Quantum Experience". In: (Dec. 2017).

[21] *Collision problem*. URL: https://en.wikipedia.org/wiki/Collision_problem (visited on).

[22] Z. Sakhi, R. Kabil, A. Tragha, and M. Bennai. "Quantum cryptography based on Grover's algorithm". In: *Second International Conference on the Innovative Computing Technology (INTECH 2012)*. IEEE, Sept. 2012. DOI: 10.1109/intech.2012.6457788. URL: http://dx.doi.org/10.1109/INTECH.2012.6457788.

[23] Pedro J Salas. "Noise effect on Grover algorithm". In: *The European Physical Journal D* 46.2 (2008), pp. 365–373.

[24] Feng Lu and Dan C. Marinescu. "Quantum Error Correction of Time-Correlated Errors". In: *School of Electrical Engineering and Computer, Science University of Central Florida* (2008).

[25] Burt Kaliski. "The mathematics of the RSA public-key cryptosystem". In: ().

[26] Christophe Pittet. "Mathematical aspects of Shor's algorithm". PhD thesis. Fourier analysis of groups in combinatorics, 2013.