SOLVING CHROMATIC NUMBER WITH QUANTUM SEARCH AND QUANTUM COUNTING

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Science

by

David Lutze

June 2021

© 2021
David Lutze
ALL RIGHTS RESERVED

COMMITTEE MEMBERSHIP

TITLE:	Solving Chromatic Number with Quantum Search and Quantum Counting
AUTHOR:	David Lutze
DATE SUBMITTED:	June 2021
COMMITTEE CHAIR:	Katharina Gillen, Ph.D. Professor of Physics
COMMITTEE MEMBER:	Theresa Migler, Ph.D. Professor of Computer Science
COMMITTEE MEMBER:	Christian Eckhardt, Ph.D. Professor of Computer Science

ABSTRACT

Solving Chromatic Number with Quantum Search and Quantum Counting

David Lutze

This thesis presents a novel quantum algorithm that solves the Chromatic Number problem. Complexity analysis of this algorithm revealed a run time of $O(\sqrt{2^n}n^2(log_2 n)^2)$. This is an improvement over the best known algorithm, with a run time of $2^n n^{O(1)}$ [1]. This algorithm uses the Quantum Search algorithm (often called Grover's Algorithm), and the Quantum Counting algorithm. Chromatic Number is an example of an **NP**-Hard problem, which suggests that other **NP**-Hard problems can also benefit from a speed-up provided by quantum technology. This has wide implications as many real world problems can be framed as **NP**-Hard problems, so any speed-up in the solution of these problems is highly sought after. A bulk of this thesis consists of a review of the underlying principles of quantum mechanics and quantum computing, building to the Quantum Search and Quantum Counting algorithms. The review is written with the assumption that the reader has no prior knowledge on quantum computing. This culminates with a presentation of algorithms for generating the quantum circuits required to solve K-Coloring and Chromatic Number.

ACKNOWLEDGMENTS

First off, thanks to Andrew Guenther, for uploading this template. Second I would like to thank my amazing advisor Katharina Gillen for being my guide and diving into the quantum computing world with me. This work would not have been possible without her. I would also like to thank my roommate Braden Lockwood for helping me make some of the Bloch Sphere figures.

We acknowledge the use of IBM Quantum services for this work. The views expressed are those of the authors, and do not reflect the official policy or position of IBM or the IBM Quantum team.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

The field of computing came about as a method to quickly solve real world problems. Over the past 70+ years, these computers have steadily become faster and more powerful, allowing for applications in just about every facet of our lives. However, now as the size of a transistor is nearing the size of individual atoms [2], the exponential growth predicted by Moore's Law may come to a standstill. There are a variety of avenues for how to address this issue, and one such avenue is Quantum Computing. It is worth noting, that quantum computers will never outright replace classical computers. Quantum computers are only faster than classical computers at certain tasks, so we will likely see a hybrid model of quantum and classical computing chips.

The general idea to utilize quantum mechanics to process and gain information was first discussed and proposed in the 1970s [3, 4]. In 1980, Paul Beinhoff introduced, the first fully described model of a computer based on quantum mechanics [5]. Then in 1981, Richard Feynman (often considered to be the father of quantum computing) first introduced the concept of quantum computing in a speech titled "Simulating Physics with Computers" [6]. This laid the fundamental foundation for quantum computing, and many regard this as the start of the field. For the next decade or so, further work was done on the foundations of quantum computing, but it was still a somewhat niche school of thought. This changed in 1994, when Peter Shor introduced his famous algorithm for factoring large integers [7]. This was a significant moment, because it essentially "broke" cryptography. Modern security systems use the Rivest, Shamir, and Adleman technique (RSA) to encrypt their data, and RSA relies on the fact that it takes a classical computer an intractable amount of time to factor

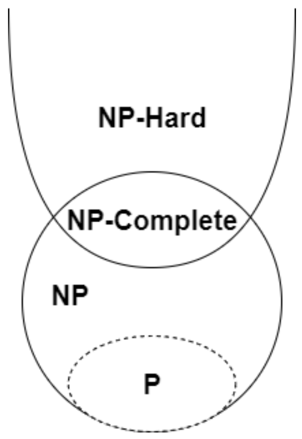


Figure 1.1: Overlap of Complexity Classes assuming P $\neq$ NP

large numbers. The computational complexity of the best known classical algorithm to solve number factoring is $O(e^{1.9(logN)^{1/3}(loglogN)^{2/3}})$ [8], but for Shor's algorithm the complexity is $O((logN)^2(loglogN)(logloglogN)$. This means that with a large enough quantum computer, someone could read *all* the private data being sent over the internet. Another important algorithm was proposed in 1996 by Lov Grover; a quantum database search [9]. This algorithm has a more modest speed up than Shor's, only quadratic instead of exponential. However Grover's algorithm makes up for it in the fact that it is incredibly general and can be applied to just about any problem.

Complexity theory in computing is used to describe certain sets of problems based on the time that it takes to solve them. Some of the most notorious problems are NP-Complete problems, which are highly prevalent in the world. Additionally, there are no known polynomial time algorithms to solve any of these problems, so any additional speed up is greatly beneficial. Since it has been shown [10] that all NP-Complete problems are reducible to each other, speeding up the solution of one will speed up the solutions of all. In this thesis, the problem we are examining is actually in NP-hard, of which NP-Complete is a subset. Figure 1.1 shows the relationship between the complexity classes.

The purpose of this thesis, is to provide a general framework for determining chromatic number of a graph, and finding the exact coloring. This framework is useful because it allows individuals who are not deeply knowledgeable of quantum computers to still reap the benefits that they offer. Additionally, the chromatic number problem is an important one to solve quickly because it has a variety of real world applications. Its most common use case is in scheduling, but it can also be used in mobile radio frequency assignment. Other applications include register allocation in CPU and compiler optimizations, and the solution of Sudoku.

The techniques presented in this thesis utilize a pair of quantum algorithms, Grover's Search Algorithm, and Quantum Counting, to solve the optimization problem of Chromatic Number in time $O(\sqrt{2^n}n^2(log_2 n)^2)$. While the problem addressed in this thesis is only one of many optimization problems, the techniques presented in this thesis can be applied to any decision problem that has an optimization version.

Chapter 2

BACKGROUND

The following sections provide background on the two main topics that this thesis pertains to; quantum computing, and complexity theory. The quantum computing background assumes that the reader has no previous knowledge on quantum computing, but is familiar with computer science, and linear algebra.

2.1 Quantum Computing

This section starts with the simplest concept in quantum computing; the quantum bit, or "qubit". From there we move to gates and circuits, and finally describe the algorithms that are key to this paper; Grover's algorithm, and Quantum Counting. The majority of the information in the following sections was adapted from [11] and [12], for more information on these concepts, please see those references.

2.1.1 Motivation

The core idea behind quantum computing is to leverage the laws of quantum mechanics in order to process information. This is in contrast to classical computing, which leverages the laws of classical mechanics in order to process information. This may seem like unnecessarily complicating things, but incorporating quantum mechanics into computation allows us to solve problems that are intractable for classical computers. This is because of two key concepts called *superposition*, and *entanglement*, which will be explained further on.

2.1.2 Single Qubit

A qubit is the most basic piece of information used by a quantum computer. Qubits are similar to classical bits in that they can be in a 0 state, or a 1 state. However what makes qubits special, is that they are described by fundamentally a different set of rules than classical bits. In a classical computer, the bit is the smallest piece of logical information available, and is represented physically by either the flow of electrons (1), or the absence of flow (0). This is accomplished through the use of transistors, which either allows energy to flow, or blocks the flow. It is clear to see that classical bits can only ever be explicitly a 1 or a 0, since electrons cannot be both flowing and not flowing at the same time. Qubits, on the other hand, play by an entirely different set of rules. Qubits are described by a wave function $|\psi\rangle$ with two basis states $|0\rangle$ and $|1\rangle$. The '$|\ \rangle$' notation, called the *Dirac notation*, is the standard notation for states in quantum mechanics, and is used throughout this paper. This is also often called the "bra-ket" notation, where $\langle\ |$ is the "bra", and $|\ \rangle$ is the "ket". This wave function is described mathematically by

$$|\psi\rangle = a\,|0\rangle + b\,|1\rangle \tag{2.1}$$

The amplitudes, a, and b can be any complex number as long as they abide by the following rule:

$$|a|^2 + |b|^2 = 1 \tag{2.2}$$

$$a, b \in \mathbb{C}$$

The rule in equation 2.2 exists because the probabilities of all the possible states must add up to 1. In other words, our qubit must have a 100% chance of being in *any* state. This means that our qubit can be in a *superposition* of both the $|0\rangle$ and $|1\rangle$

basis states, so it can be both 0 and 1 at the same time! However this superposition is only present when the quantum state $|\psi\rangle$ is *unobserved*. When we measure the qubit (i.e. observe it), it must collapse to one of the two basis states. The probability of the qubit collapsing in either state is described by the amplitudes a and b. Specifically, the probability of observing $|0\rangle$ when we measure our state $|\psi\rangle$ is $|a|^2$. If we were to describe our classical bits using the same notation, we could do so with the following system of equations:

$$|\psi_c\rangle = x\,|0\rangle + y\,|1\rangle \tag{2.3}$$

$$x, y \in \{0, 1\}, \quad y = \bar{x} \tag{2.4}$$

In equation 2.4 we see that x and y can only be 0 or 1, and that y must be the opposite of x. This encapsulates the fact the our classical bit $|\psi_c\rangle$ can only ever be exactly 0 or 1, regardless of whether we are observing it or not.

A wonderful visualization of a single qubit exists in the form of a Bloch Sphere (named after Felix Bloch). But before we can represent our qubit with a Bloch Sphere, we must redefine the amplitudes a and b from equation 2.2 into a form that can be plotted on a sphere. Instead of letting a and b be complex, we define them to be real numbers, and introduce a relative phase term, which tells us the phase difference between the two states:

$$|\psi\rangle = a\,|0\rangle + e^{i\phi} b\,|1\rangle \tag{2.5}$$

$$a, b, \phi \in \mathbb{R}$$

In order to understand how the above equation can still fully describe a single qubit state, we need to briefly discuss a concept called "global phase". The global phase of a state is a phase shift that is applied to every component of the overall state. For example, imagine we have a qubit in the state $|\psi\rangle = i\,|1\rangle$. Notice that this qubit has a phase i. If we measure this state, we will simply measure the state $|1\rangle$. Because of this, we can say that the states $i\,|1\rangle$ and $|1\rangle$ are physically equivalent. Measuring them will both yield the state $|1\rangle$. Because of this we can safely ignore this global phase.

With this representation, we are closer to being able to meaningfully visualize our qubit. The next step that we take is to apply the following trig identity:

$$cos^2x + sin^2x = 1 \tag{2.6}$$

This should look familiar, as it is of the same form as equation 2.2. Using this, we can redefine our amplitudes in the following way:

$$a = \cos\frac{\theta}{2}, \quad b = \sin\frac{\theta}{2} \tag{2.7}$$

From here, we can fully describe the state of any qubit with two variables, θ and ϕ:

$$|\psi\rangle = \cos\frac{\theta}{2}\,|0\rangle + e^{i\phi}\sin\frac{\theta}{2}\,|1\rangle \tag{2.8}$$

$$\theta, \phi \in \mathbb{R}$$

It may seem odd why we chose $\frac{\theta}{2}$ instead of just θ, but the reason is because the θ we chose, directly corresponds to the angle from the z axis in our Bloch Sphere representation. This can be seen in figure 2.1

A Bloch Sphere is a essentially a special unit sphere. The qubit itself is described by a unit vector, which can reach any point on the surface of the sphere. Typically, we define the north pole of the sphere to represent the state $|0\rangle$, and the south pole of the sphere to represent the $|1\rangle$ state. Examples of common states, represented on Bloch Spheres can be seen in figure 2.2. Note that this figure and all similar Bloch Sphere

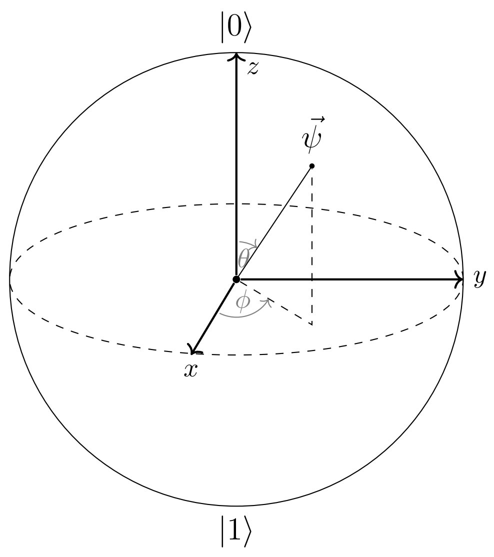


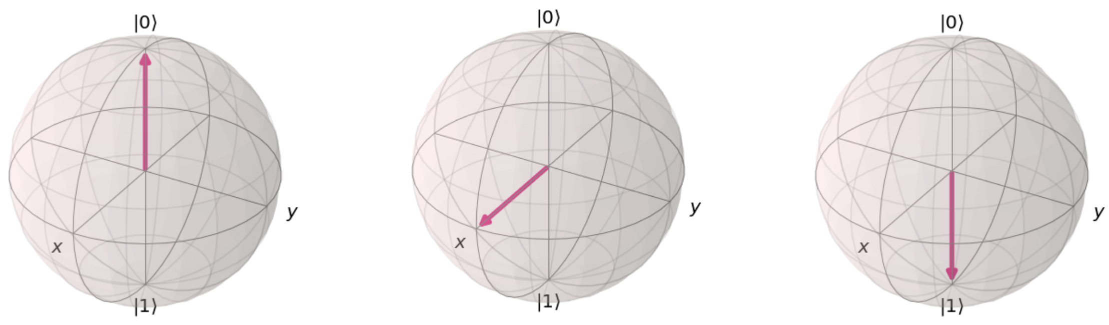
Figure 2.1: General Bloch Sphere with angles θ and ϕ defined

figures were created using Qiskit [12]. Figure 2.2b shows a common state, which is often called the equal superposition state. When measuring this state, there is a 50% chance that the measurement will yield $|0\rangle$, and a 50% chance that the measurement will yield $|1\rangle$. The $\frac{1}{\sqrt{2}}$ in the equal superposition state is called the normalizing factor, and that ensures we are abiding by the rule in equation 2.2. We can verify that this is true with the following:

$$\left|\frac{1}{\sqrt{2}}\right|^2 + \left|\frac{1}{\sqrt{2}}\right|^2 = \frac{1}{2} + \frac{1}{2} = 1 \tag{2.9}$$

When describing qubits, we often use a vector to describe the amplitudes of the state, since this is the key information that will change as we perform computations on our qubit. The vectors representing the states $|\psi\rangle = |0\rangle$, and $|\psi\rangle = |1\rangle$ are shown below:

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{2.10}$$



(a) $|\psi\rangle = |0\rangle$ (b) $|\psi\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ (c) $|\psi\rangle = |1\rangle$

Figure 2.2: Bloch Spheres with vectors representing three different qubit states

We see that the value in the first row of our state vector corresponds to the probability amplitude of state $|0\rangle$, and the value in the second row corresponds to the probability amplitude of state $|1\rangle$. In the general case, we can describe the state $|\psi\rangle = a|0\rangle + b|1\rangle$ with the vector

$$\begin{bmatrix} a \\ b \end{bmatrix} \begin{matrix} \leftarrow |0\rangle \\ \leftarrow |1\rangle \end{matrix} \tag{2.11}$$

This is the same idea as when we write a spatial vector as a combination of $\hat{\imath}$ and $\hat{\jmath}$ unit vectors. The difference is that when describing a qubit, the $\hat{\imath}$ and $\hat{\jmath}$ are replaced with $|0\rangle$ and $|1\rangle$.

Now that we have an understanding of what a qubit is and how to describe it, we can move on to gates, which are the mechanism with which we can change the states of qubits.

2.1.3 Measurement

Measuring qubit states is how we obtain information about them. Measuring quantum states is determined by probabilities. The probability of measuring the state $|\psi\rangle$ in the basis state $|x\rangle$ is found by

$$p(|x\rangle) = |\langle x|\psi\rangle|^2 \tag{2.12}$$

From section 2.1.2, we saw that the $|\ \rangle$ represents a column vector. Similarly, the $\langle\ |$ represents a row vector. We can translate between the two using the conjugate transpose, i.e. in order to get $\langle\psi|$ from $|\psi\rangle$, we transpose $|\psi\rangle$ and then take the complex conjugate (change the sign of all imaginary numbers).

Lets imagine that we have the following state:

$$|\psi_0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \tag{2.13}$$

and we want to find the probability of measuring state $|0\rangle$. In order to do so, we apply equation 2.12

$$p(|0\rangle) = |\langle 0|\psi_0\rangle|^2 = \left| \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix} \right|^2 = \left| \frac{1}{\sqrt{2}} \right|^2 = \frac{1}{2} \tag{2.14}$$

As expected, we see that the probability of measuring $|\psi_0\rangle$ in the state $|0\rangle$ is 1/2.

2.1.4 Single Qubit Gates

Gates are the way that we manipulate information in a computer. This is true for both classical, and quantum computers. Gates are of the utmost importance, because without them we would have no way to change the information contained in our bits or qubits. Without gates, computers wouldn't be able to solve any problems.

Table 2.1: Truth table for classical NOT gate

Input	Output
0	1
1	0

In a classical computer, there is only one gate that acts on a single bit: the NOT gate. This gate simply flips the state of a bit, from a 0 to a 1, and from a 1 to a 0. Table 2.1 shows the truth table for a NOT gate.

In quantum computers, on the other hand, there are actually infinitely many single qubit gates. As we saw in section 2.1.2, a qubit can be represented as any point along the surface of the Bloch Sphere. A single qubit gate is an operation that essentially *moves* the qubit from one place on the Bloch Sphere, to another place on the Bloch Sphere. These single qubit gates are represented by 2x2 matrices, and the only rule is that they must be unitary:

$$U^{\dagger}U = I \tag{2.15}$$

Here $U^{\dagger}$ is the *adjoint* of U, and is obtained by transposing U, then taking the complex conjugate of the transposed U. I represents the two by two identity matrix.

While there are an infinite number of matrices that satisfy this rule, in practice, we only use a handful of single qubit gates. Three of the most common gates, are represented by the Pauli matrices, and are called the X, Y, and Z gates.

The first gate we will look at is the X-gate. The X-gate is defined by the following matrix:

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \tag{2.16}$$

The circuit diagram for an X-gate is shown in figure 2.3. If this format is unfamiliar, that's okay, quantum circuits are explained in detail in section 2.1.8. Note that all

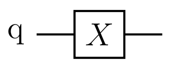


Figure 2.3: Circuit diagram for X-gate

circuit diagrams were created with [13]. Additionally, a compilation of all qubit gates referenced in this work are in the Appendix.

This is the quantum equivalent of the classical NOT gate. In order to see why this is the case, we can apply the X-gate to a qubit in the $|0\rangle$ state. In order to apply a gate to a qubit, we simply multiply the qubits' state vector by the gate's matrix. We recall from equation 2.10 what the vector representations of $|0\rangle$ and $|1\rangle$ are.

$$X|0\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = |1\rangle \tag{2.17}$$

When applying the X-gate to the general qubit state $|\psi\rangle = a|0\rangle + b|1\rangle$, we can see that the action it performs is *swapping* the amplitudes.

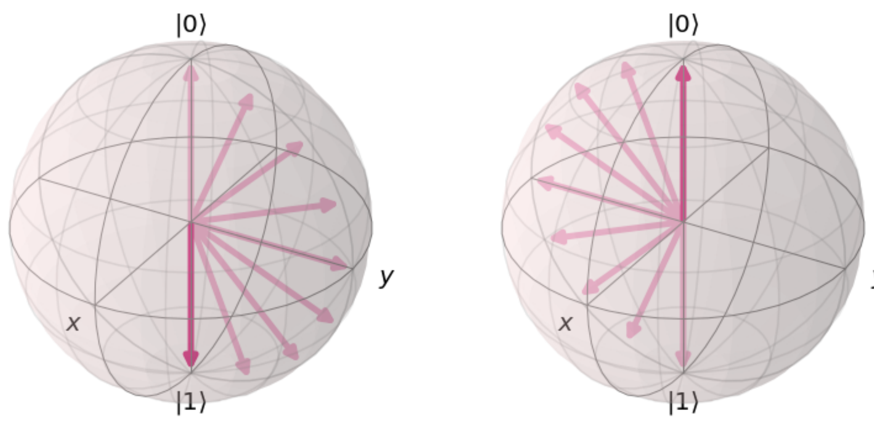
$$X|\psi\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} b \\ a \end{bmatrix} = b|0\rangle + a|1\rangle \tag{2.18}$$

Returning to our view of the Bloch Sphere, we can see that the X-gate, corresponds to a π rotation around the x-axis. 2.4.

The next gate is the Y-gate. The Y-gate is defined by the following matrix:

$$Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \tag{2.19}$$

The circuit diagram for the Y-gate is shown in figure 2.5.



(a) $X\,|0\rangle$ (b) $X\,|1\rangle$

Figure 2.4: Bloch Sphere visualizations of applying X-gate to single qubit

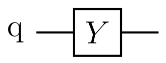


Figure 2.5: Circuit diagram for Y-gate

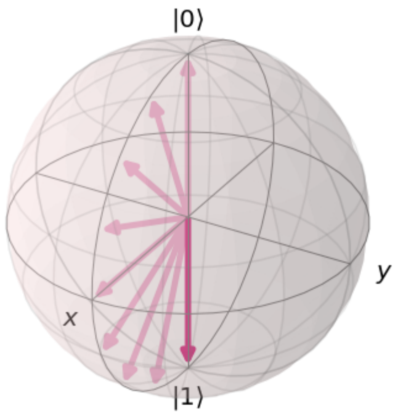
This is similar to the X-gate in that it swaps the probability amplitudes, but it also adds a phase shift. Since this phase shift does not change anything when we measure the qubit, the Y-gate is not used very often.

Analogous to the X-gate, we can visualize the action of the Y-gate as a π rotation around the y-axis of the Bloch Sphere. This is shown in figure 2.6

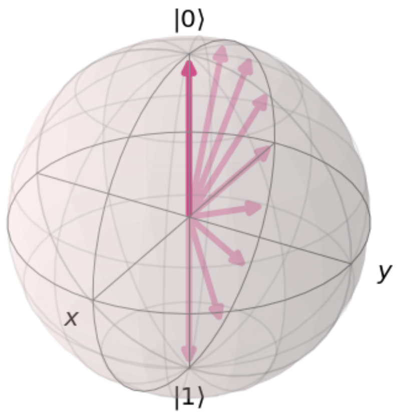
Finally, we come to the Z-gate. The Z-gate is defined as follows:

$$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \tag{2.20}$$

The circuit diagram for the Z-gate is shown in figure 2.7.



(a) $Y|0\rangle$



(b) $Y|1\rangle$

Figure 2.6: Bloch Sphere visualizations of applying Y-gate to single qubit

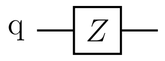


Figure 2.7: Circuit diagram for Z-gate

Applying to our general state, $|\psi\rangle = a|0\rangle + b|1\rangle$, we can see that the Z-gate adds a negative phase to the $|1\rangle$ piece of our qubit:

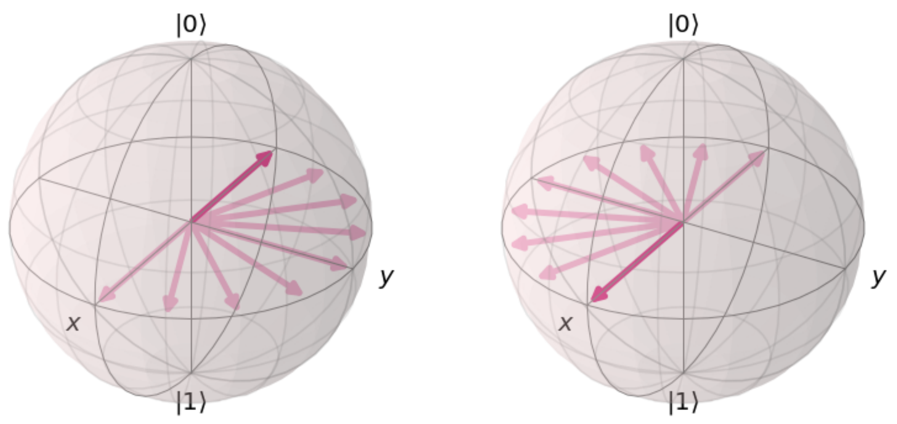
$$Z|\psi\rangle = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} a \\ -b \end{bmatrix} = a|0\rangle - b|1\rangle \tag{2.21}$$

As with the previous two gates, we can visualize the action of the Z-gate as a rotation of π around the z-axis of the Bloch Sphere. This is shown in figure 2.8

The final single qubit gate we will examine is the Hadamard gate, or H-gate. This is one of the most useful single qubit gates, because it does what none of the other previous gates have done; it puts a qubit in a *superposition*. The H-gate is defined as follows:

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{2.22}$$

The circuit diagram for the H-gate is shown in figure 2.9. We can immediately notice



(a) $Z\left|+\right\rangle$ (b) $Z\left|-\right\rangle$

Figure 2.8: Bloch Sphere visualizations of applying Z-gate to single qubit

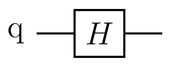


Figure 2.9: Circuit diagram for H-gate

that this gate is different from the others in two ways. It has no elements that are 0, and it is multiplied by a factor of $\frac{1}{\sqrt{2}}$. This $\frac{1}{\sqrt{2}}$ is called the normalization factor, and is required in order to satisfy the rule in equation 2.15. In order to see why the H-gate is so special, we will first apply it to the $\left|\psi\right\rangle = \left|0\right\rangle$ state.

$$H\left|0\right\rangle = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\1 & -1\end{bmatrix}\begin{bmatrix}1\\0\end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\1\end{bmatrix} = \frac{1}{\sqrt{2}}(\left|0\right\rangle + \left|1\right\rangle) \tag{2.23}$$

We can see that when applying the H-gate to the $\left|0\right\rangle$ state, we got out a state in equal superposition. If we were to measure this new state, we would have a 50% chance of seeing a $\left|0\right\rangle$ and a 50% chance of seeing a $\left|1\right\rangle$. The state $\frac{1}{\sqrt{2}}(\left|0\right\rangle + \left|1\right\rangle)$ is often denoted as $\left|+\right\rangle$. Another useful property of the H-gate reveals itself when we apply it a second time to our $\left|+\right\rangle$ state:

$$H\left|+\right\rangle = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\1 & -1\end{bmatrix}\frac{1}{\sqrt{2}}\begin{bmatrix}1\\1\end{bmatrix} = \frac{1}{\sqrt{2}} \times \frac{1}{\sqrt{2}}\begin{bmatrix}1+1\\1-1\end{bmatrix} = \frac{1}{2}\begin{bmatrix}2\\0\end{bmatrix} = \begin{bmatrix}1\\0\end{bmatrix} = \left|0\right\rangle \tag{2.24}$$

We are now back to our original state! This is because the H-gate is its' own complex conjugate:

$$H^\dagger H = HH = I \tag{2.25}$$

This is actually true of the X, Y, and Z gates as well, and it is part of the reason they are heavily used in quantum algorithms.

The other useful state that the H-gate produces, is found when we apply the H-gate to the $|1\rangle$ state:

$$H|1\rangle = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}\begin{bmatrix} 0 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ -1 \end{bmatrix} = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \tag{2.26}$$

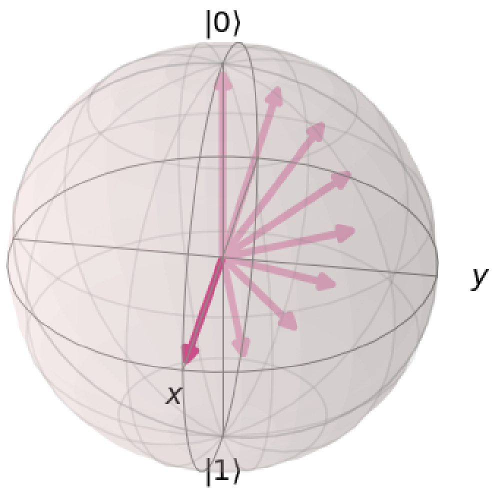
This is another equal superposition state, but this time we have a phase difference. This state is commonly denoted as $|-\rangle$. As with the $|+\rangle$ state, we can see that applying the H-gate to the $|-\rangle$ state, returns it to its' original state:

$$H|-\rangle = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}\frac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ -1 \end{bmatrix} = \frac{1}{\sqrt{2}} \times \frac{1}{\sqrt{2}}\begin{bmatrix} 1-1 \\ 1+1 \end{bmatrix} = \frac{1}{2}\begin{bmatrix} 0 \\ 2 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = |1\rangle \tag{2.27}$$

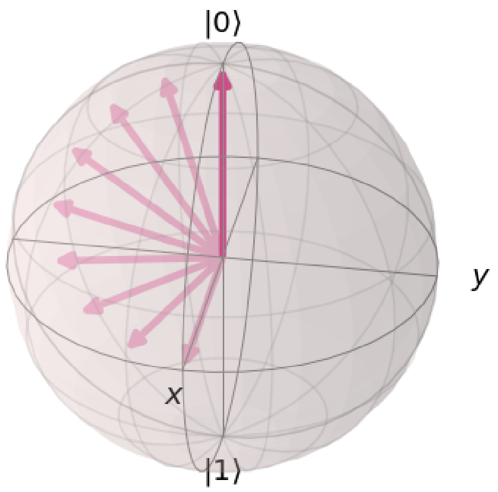
Finally, we will look at what happens when we apply the H-gate to the general state $|\psi\rangle = a|0\rangle + b|1\rangle$

$$H(a|0\rangle + b|0\rangle) = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}\begin{bmatrix} a \\ b \end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix} a+b \\ a-b \end{bmatrix} = \frac{1}{\sqrt{2}}\Big((a+b)|0\rangle + (a-b)|1\rangle\Big) \tag{2.28}$$

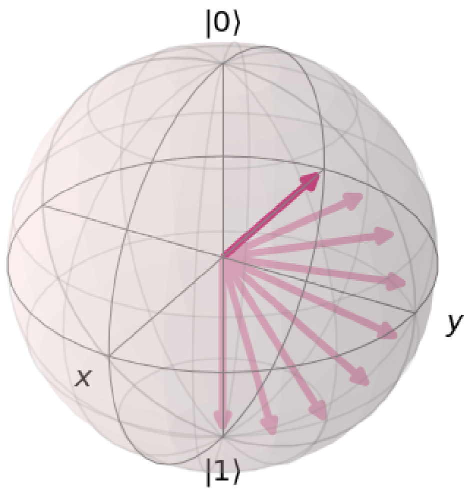
The H-gate can also be visualized on the Bloch Sphere as a rotation of π around the line between the x and z-axis. This is shown in figure 2.10.



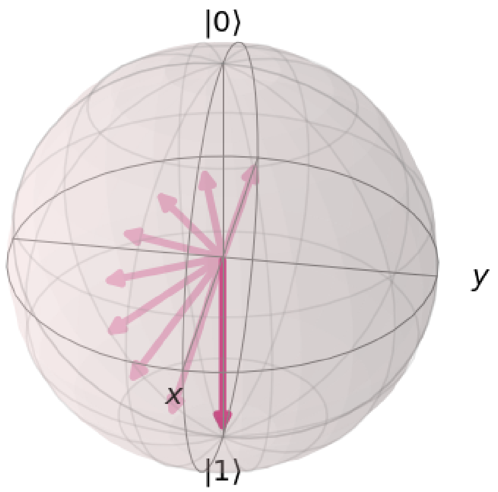
(a) $H\,|0\rangle$



(b) $H\,|+\rangle$



(c) $H\,|1\rangle$



(d) $H\,|-\rangle$

Figure 2.10: Bloch Sphere visualizations of applying H-gate to single qubit

Now that we understand what a single qubit is, and how gates can act on single qubit states, we can move on to states with more than one qubit.

2.1.5 Two Qubits

In a system with two qubits, as with a system of two bits, we now have four different basis states. We can write our system as a tensor product of the two different qubits. Adopting the notation $|0\rangle_i$, we let i denote which qubit we are talking about. Using this notation, a two qubit system can be written as

$$|\psi\rangle = a\,|0\rangle_0 \otimes |0\rangle_1 + b\,|0\rangle_0 \otimes |1\rangle_1 + c\,|1\rangle_0 \otimes |0\rangle_1 + d\,|1\rangle_0 \otimes |1\rangle_1 \tag{2.29}$$

Often the tensor product is omitted, as it is implied that two $|\ \rangle$'s next to each other are tensor multiplied together. Thus we would write equation 2.29 as:

$$|\psi\rangle = a\,|0\rangle_0\,|0\rangle_1 + b\,|0\rangle_0\,|1\rangle_1 + c\,|1\rangle_0\,|0\rangle_1 + d\,|1\rangle_0\,|1\rangle_1 \tag{2.30}$$

This is the same combination that we would see with two classical bits. Note that the qubits are numbered starting at 0, which is the convention used throughout this paper. Usually we would not be explicit with the numbering of each qubit, and represent the above state as follows:

$$|\psi\rangle = a\,|00\rangle + b\,|01\rangle + c\,|10\rangle + d\,|11\rangle \tag{2.31}$$

It is also worth introducing another piece of notation here:

$$|\psi\rangle = |0\rangle^{\otimes 2} \tag{2.32}$$

Here we see that the state $|0\rangle$ is raised to the power of $\otimes 2$. What this means is we actually have two $|0\rangle$ states tensor multiplied together:

$$|\psi\rangle = |0\rangle^{\otimes 2} = |0\rangle \otimes |0\rangle = |00\rangle \tag{2.33}$$

Similar to our single qubit state, the following restriction is placed on the amplitudes a, b, c and d:

$$|a|^2 + |b|^2 + |c|^2 + |d|^2 = 1 \tag{2.34}$$

$$a, b, c, d \in \mathbb{C}$$

The number of amplitudes in each state, grows according to 2^n where n is the number of qubits. Additionally, the sum of squared amplitudes always needs to equal 1 (as shown in equations 2.2 and 2.34). This is because the probability of the state $|\psi\rangle$ being in one of the four basis states must be 1. As with our single qubit state, we often represent the two qubit state as an amplitude vector:

$$\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} \begin{array}{l} \leftarrow |00\rangle \\ \leftarrow |01\rangle \\ \leftarrow |10\rangle \\ \leftarrow |11\rangle \end{array} \tag{2.35}$$

The amplitudes are ordered such that a is the amplitude of the $|00\rangle$ state, and d is the amplitude of the $|11\rangle$ state, as shown in equation 2.35.

Looking back at equation 2.30, we observe that each qubit is represented in its own '$|\ \rangle$', whereas equation 2.31 shows both qubits in a single '$|\ \rangle$'. These equations are equal, but in equation 2.30 there is actually an implied tensor operator '$\otimes$' between each qubit. The full equation is written as

$$|\psi\rangle = a(|0\rangle_0 \otimes |0\rangle_1) + b(|0\rangle_0 \otimes |1\rangle_1) + c(|1\rangle_0 \otimes |0\rangle_1) + d(|1\rangle_0 \otimes |1\rangle_1) \tag{2.36}$$

The tensor product was omitted from equation 2.30 for simplicity's sake, but any time we see two '$|\ \rangle$'s next to each other we can assume they are being tensor multiplied together. This tensor product is how we get from a vector of length two that represents a single qubit, to a vector of length four that represents a two qubit state (equation 2.35). In order to see how this is true, we can expand the expression $a(|0\rangle_0 \otimes |0\rangle_1)$:

$$a(|0\rangle_0 \otimes |0\rangle_1) = a\left(\begin{bmatrix}1\\0\end{bmatrix} \otimes \begin{bmatrix}1\\0\end{bmatrix}\right) = a\begin{bmatrix}1 \times \begin{bmatrix}1\\0\end{bmatrix}\\ 0 \times \begin{bmatrix}1\\0\end{bmatrix}\end{bmatrix} = a\begin{bmatrix}1\\0\\0\\0\end{bmatrix} = \begin{bmatrix}a\\0\\0\\0\end{bmatrix} = a\,|00\rangle \tag{2.37}$$

We can use the tensor product to describe states with arbitrarily many qubits. Additionally, we can observe that as the number of qubits grows, the number of variables used to describe the state grows exponentially. We saw that for a single qubit, its state vector contains two variables. For a two qubit state, the state vector contains four variables. If we let n denote the number of qubits in our state, then the number of variables in that state is 2^n. This is one of the key draws of quantum computing as it vastly cuts down on the physical space required to represent vast amounts of information. In fact, a quantum computer with only 300 qubits, contains as many variables as there are atoms in the observable universe! However it is important to note that there is no easy way to extract the exact values of these variables. This is because measuring the state collapses it to a single value, which does not tell us about the amplitude variables. The best we could do is repeatedly prepare and measure a state and try to piece together the actual amplitude values based on how often we measured certain states.

Unfortunately, a representation like the Bloch Sphere does not exist for two qubit states, as we would need to plot it in four dimensional Hilbert space, which would be quite hard to visualize. Now that we understand how to describe states with two or more qubits, we can look at gates that act on two qubits.

2.1.6 Multi Qubit Gates

In this section, we will explore some of the most commonly used two qubit gates. As with our single qubit gates, the two qubit gates are also represented by a matrix, but this time they are 4×4 matrices. This rule extends to gates acting on a state with n qubits, such that the size of the matrix will always be $N \times N$, where $N = 2^n$. Before exploring gates that explicitly act on two qubits, we will examine how to apply single qubit gates on multi-qubit states.

If we have two qubits in our state, any single qubit gate must also be described by a 4x4 matrix. In order to determine the values of the extended gate, we use the tensor product. For example, imagine we have the following two qubit state $|\psi\rangle = |00\rangle$, and we want to apply the Hadamard gate to the first qubit. We can represent this as an H-gate applied to the first qubit tensor-multiplied with the Identity matrix applied to the second qubit:

$$H|0\rangle_0 \otimes I|0\rangle_1 = (H \otimes I)|00\rangle \tag{2.38}$$

Calculating the tensor product $H \otimes I$, we get the following:

$$\begin{aligned} H \otimes I = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} &= \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \times \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & 1 \times \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ 1 \times \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & -1 \times \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \end{bmatrix} \\ &= \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix} \end{aligned} \tag{2.39}$$

If we apply this to our state vector for $|\psi\rangle = |00\rangle$, we get:

$$\begin{aligned} (H \otimes I) |00\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} &= \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} \\ = \frac{1}{\sqrt{2}}(|00\rangle + |10\rangle) = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) |0\rangle &= |+\rangle |0\rangle = |+0\rangle \end{aligned} \tag{2.40}$$

Here we get what we would expect, which is the first qubit in the $|+\rangle$ state, and the second qubit unaltered. Using the tensor product, we can expand our gate applications to any number of qubits with any combination of gates. For example, the two qubit gate which applies the H-gate to both qubits can be found with $H \otimes H$.

Now we will explore two commonly used gates that are explicitly built for operating on two qubits. These are the controlled PHASE gate, and the controlled NOT gate. The controlled PHASE gate (often abbreviated as cPHASE) is defined by the following

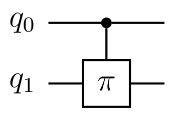


Figure 2.11: Circuit diagram for cPHASE gate

matrix:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\phi} \end{bmatrix} \tag{2.41}$$

The circuit diagram for the cPHASE is shown in figure 2.11.

The first thing that we note is that this gate has a parameter ϕ, which ranges from 0 to 2π, and allows us to control the phase applied by this gate. We also note that this gate only applies this phase to the $|11\rangle$ basis state:

$$(cPHASE)\,|\psi\rangle = a\,|00\rangle + b\,|01\rangle + c\,|10\rangle + e^{i\phi}d\,|11\rangle \tag{2.42}$$

The most common form of the cPHASE gate, is the π cPHASE gate, which results in adding a negative sign to the $|11\rangle$ state:

$$e^{i\pi}\,|11\rangle = -\,|11\rangle \tag{2.43}$$

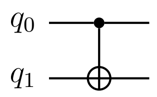


Figure 2.12: Circuit diagram for cNOT gate

Table 2.2: Truth table for controlled NOT gate. Qubit 0 is control and qubit 1 is target

Input	Output
00	00
01	01
10	11
11	10

The next gate we will look at is the controlled NOT gate, sometimes written as cNOT or CX. The cNOT gate is described by the following matrix:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{2.44}$$

The circuit diagram for the cNOT is shown in figure 2.12. With this gate, the first qubit is often called the *control* qubit, and the second qubit the *target* qubit. If the control qubit is a 0, then the target qubit is left unchanged. However if the control qubit is a 1, then the target qubit is NOT'd. This is shown in table 2.2.

We can see that the cNOT gate left the states $|00\rangle$ and $|01\rangle$ unchanged. For the state $|10\rangle$, it flipped the second qubit to a 1, resulting in the state $|11\rangle$. And for the state $|11\rangle$ it flipped the second qubit to a 0, resulting in the state $|10\rangle$. If we apply it to the general state $|\psi\rangle = a|00\rangle + b|01\rangle + c|10\rangle + d|11\rangle$, we see that it swaps the amplitudes

Table 2.3: Truth table for controlled NOT gate. Qubit 1 is control and qubit 0 is target

Input	Output
00	00
01	11
10	10
11	01

of the $|10\rangle$ and $|11\rangle$ states:

$$(cNOT)\,|\psi\rangle = a\,|00\rangle + b\,|01\rangle + d\,|10\rangle + c\,|11\rangle \tag{2.45}$$

It is worth noting that the above discussion of the cNOT gate is for the case when qubit 0 is the control qubit, and qubit 1 is the target qubit. When we want qubit 0 to be the target qubit, and qubit 1 to be the control qubit, the cNOT gate is defined as follows:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \tag{2.46}$$

The truth table for this version of the cNOT gate is shown in table 2.3

The last gate I will introduce, is a commonly used multi-qubit gate; the Toffoli gate. The Toffoli gate is essentially an extension of the two-qubit cNOT gate, except that it has multiple control qubits. It is often also called the controlled-controlled-NOT

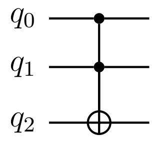


Figure 2.13: Circuit diagram of Toffoli gate. q_0 and q_1 are the controls, and q_2 is the target.

gate, or CCX gate. The matrix for the Toffoli gate is shown below:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \tag{2.47}$$

A circuit representation of the Toffoli gate is shown in figure 2.13. Here we can see that qubits q_0 and q_1 are the control qubits, and qubit q_2 is the target. If qubits q_0 and q_1 are both in the $|1\rangle$ state, then q_2 will be flipped. Another way to represent this, is that after the Toffoli gate, the value of q_2 is $q_2 \bigoplus q_0 \cdot q_1$.

The idea of a Toffoli gate can be extended to a gate with an arbitrary number of controls, and a single target. These gates are often called multi-controlled-Toffoli gates (MCT), or multi-controlled-NOT gates (MCX). This concept of multi-controlled gates extends beyond just X gates, to any unitary operation U. In fact, we can also control U operators that act on more than one qubit (we will take advantage of this later on). The representation stays the same, with the controls being connected by a straight line and single dots, to the gate they are controlling. This is shown in figure 2.14

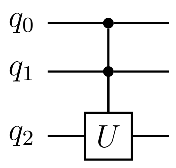


Figure 2.14: Circuit diagram of controlling a general gate U. q_0 and q_1 are the controls.

While the methods for making any general gate U a controlled gate are outside the scope if this document, they are detailed in [14].

2.1.7 Entanglement

The concept of entanglement is another key aspect that separates quantum computing from classical computing. The basic idea of entanglement is that manipulating one qubit can change the state of another qubit [11, 12]. A great example of this is measurement. Think about a classical string of bits, x_0x_1. If we want to know the overall state of these bits, we have to look explicitly at x_0, and at x_1. If x_0 is a 0, this does not tell us anything about whether or not x_1 is a 0 or a 1.

With qubits however, there is a way for us to look at (measure) x_0 and *without* measuring x_1, know what state it is in. In order to see how this works, we will work with a special state $|\psi\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$. This is known as a Bell state. It is also an *entangled* state, meaning that qubits 0 and 1 are entangled. If we measure qubit 0 of $|\psi\rangle$, it must collapse into either a 0 or a 1. If we measure qubit 0 and we get $|0\rangle$, then without measuring qubit 1, we already know that it also is in state $|0\rangle$. One way to determine if a state is entangled, is if the individual qubits are linearly separable. Take the state $|\psi\rangle = \frac{1}{\sqrt{2}}(|01\rangle + |11\rangle)$ for example. This state can be expressed as:

$$|\psi\rangle = \frac{1}{\sqrt{2}}(|01\rangle + |11\rangle) = \frac{1}{\sqrt{2}}(|0\rangle_0 + |1\rangle_0) \otimes |1\rangle_1 \tag{2.48}$$

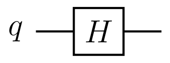


Figure 2.15: Quantum Circuit with an H-gate acting on a single qubit

Since we can separate the two qubits we know that the state is not entangled. If we measure qubit 1, we still don't know what state qubit 0 is in. The state $|\psi\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ however, is not linearly separable, and because of this we know that it is entangled.

Now that we have a basic understanding of qubits and qubit gates, we can move on to the most commonly used method for representing interactions between qubits and qubit gates; a quantum circuit.

2.1.8 Quantum Circuit

In this section, I introduce the concept of a quantum circuit diagram, and show how these are commonly drawn.

A simple example of a quantum circuit is shown in figure 2.15. This circuit contains a single qubit, labeled q. A single Hadamard gate acts on this one qubit, as shown by the box with the H inside. The circuits are read and processed from left to right. The left side of this circuit would be our input, and the right side of this circuit would be the output. If our qubit is initially in state $|\psi_0\rangle = |0\rangle$, at the end of the circuit our qubit will be in the state $|\psi_1\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$. This is because all this circuit does is apply a single H-gate to our qubit.

Figure 2.16 shows a more complex circuit. Here we have two qubits, labeled q_0 and q_1. The first thing that this circuit does, is apply the H-gate to qubit q_0. After this, it applies the cNOT gate on both qubits. The small dot on q_0 signifies that q_0 is the control, and the $\bigoplus$ symbol on q_1 signifies that q_1 is the target qubit. $\bigoplus$ represents

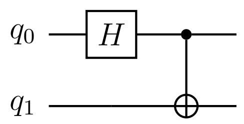


Figure 2.16: Quantum Circuit with two qubits. An H-gate and a cNOT-gate are present

addition modulo 2, and the reason it is used is because the value of q_1 after the cNOT gate will be $q_1 \bigoplus q_0$. If we start in the state $|00\rangle$, our output state is $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$. This result is explicitly worked out in equation 2.49. Here H_0 signifies that we are applying the H-gate to qubit 0.

$$\begin{aligned} cNOT(H_0\,|00\rangle) &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \times \left(\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \right) \\ &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) \end{aligned} \tag{2.49}$$

This output state is a commonly used state, and is known as a Bell state. Another way to arrive at this state, without explicitly doing the math, is to piece together the known actions of the gates on our state. Starting in the state $|\psi_0\rangle = |00\rangle$, we observe that the first action is to apply a H-gate to the qubit q_0. This will place q_0 in the equal superposition state $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and will leave q_1 untouched. Thus, after the H-gate on q_0, our system is in the state $|\psi_1\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |10\rangle)$. Notice how in this state, q_0 is in a superposition of $|0\rangle$ and $|1\rangle$, while q_1 is still in the $|0\rangle$ state. Next we apply the

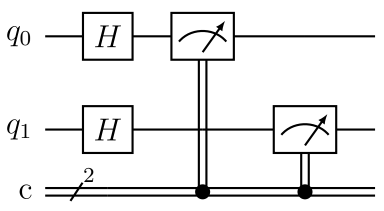
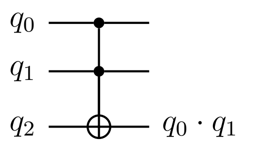


Figure 2.17: Quantum circuit with qubits and classical bits. Each qubit is measured into a corresponding classical bit

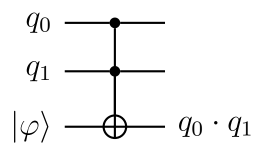
cNOT gate. From table 2.2, we can see that the state $|00\rangle$ will be unchanged, but the state $|10\rangle$ will be changed to $|11\rangle$. Thus our final state is $|\psi_2\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$.

One important aspect of quantum circuits, is the use of classical bits. Classical bits are used, because often we want to take the result of a quantum operation, and perform some classical processing on it. While quantum computers offer great solutions to some specific problems, the majority of computational work will not benefit from being done on a quantum computer. Because of this, a hybrid model is used, where a classical computer is used to prepare information for the quantum algorithm, then the classical computer uses the results from the quantum computer to complete the task at hand. Figure 2.17 shows a simple circuit with two qubits, and two classical bits. The classical bits are represented as the two parallel gray lines, and the '/2' signifies that there are two bits on that line. After the two Hadamard gates, our state will be $|\psi\rangle = \frac{1}{2}(|00\rangle + |01\rangle + |10\rangle + |11\rangle)$. The boxes connected to the classical qubits represent that the qubits are being measured, and the results are being stored in the classical bits. q_0 is stored in classical bit 0, and q_1 is stored in classical bit 1. When the qubits are measured, they can no longer be in a superposition, and must collapse to a single state. There is an equal 25% chance that we will measure any of our four basis states.

Another concept worth introducing is that of an "output qubit." Essentially, this is just a qubit that holds information related to a specific operation. For example if we



(a) Example circuit with an output qubit, q_2



(b) Example circuit with an output qubit labeled as $|\varphi\rangle$

Figure 2.18: Example circuits demonstrating how output qubits will be represented

have a MCT gate controlled by qubits q_0 and q_1, targeting q_2, we could call q_2 the output qubit. It holds the information $q_0 \cdot q_1$. This circuit is shown in figure 2.18a. In cases where I introduce circuits that have an output qubit, I will use $|\varphi\rangle$ to represent that output qubit. This is shown in figure 2.18b.

2.1.9 Uncomputation

In Quantum Computing, uncomputation is a heavily utilized technique. Basically, what uncomputation is, is reversing the computational work done to a certain set of qubits. For example, imagine I applied an X-gate to a single qubit in order to *compute* its inverse. If I then wanted to *un* compute that qubit, I need only apply another X-gate. The application of this second X-gate will return the qubit to the original state it was in before the first X-gate was applied. This process is shown in figure 2.19. The reason that uncomputation is useful, is because it reduces the amount of ancillary qubits required to implement quantum algorithms. With uncomputation, these ancillary qubits can be re-used throughout the algorithm. This is useful because current quantum computers have a very limited number of qubits available.

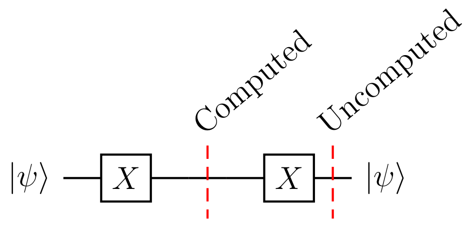


Figure 2.19: Example of Uncomputation

2.1.10 Phase Kickback

Phase kickback is an interesting effect that is used in many quantum algorithms. The basic premise is that, when performing a controlled operation, if the control qubit is in a superposition, the state of the *control* qubit is changed, and the target qubit is left untouched. In order to illustrate this, we will examine a gate called the T-gate, and its controlled version. The T-gate is described by the following matrix:

$$T = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix} \tag{2.50}$$

and the controlled T-gate is described by:

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\pi/4} \end{bmatrix} \tag{2.51}$$

To see phase kickback in action, we will apply our controlled T-gate to the state $|\psi\rangle = |{+}1\rangle$, where the first qubit ($|+\rangle$) is the control, and the second qubit ($|1\rangle$) is the target. Recall that we can represent our two qubit state as the tensor product of two single qubit states:

$$\begin{aligned} |{+1}\rangle &= \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes |1\rangle \\ &= \frac{1}{\sqrt{2}}(|01\rangle + |11\rangle) \end{aligned} \tag{2.52}$$

Then we apply the controlled T-gate:

$$CT\,|{+1}\rangle = \frac{1}{\sqrt{2}}(|01\rangle + e^{i\pi/4}\,|11\rangle) \tag{2.53}$$

The interesting thing occurs when we re-seperate the two qubits:

$$\frac{1}{\sqrt{2}}(|01\rangle + e^{i\pi/4}\,|11\rangle) = \frac{1}{\sqrt{2}}(|0\rangle + e^{i\pi/4}\,|1\rangle) \otimes |1\rangle \tag{2.54}$$

We see that the target qubit is left completely untouched, whereas the control qubit now has a relative phase shift of $e^{i\pi/4}$ applied to it. This is the core concept of phase kickback: a controlled operation, performed when the *control* qubit is in a superposition, will 'kick back' the eigenvalue added by the gate to the control qubit. An eigenvalue is a special number associated with a matrix and its eigenvector. These are defined as follows:

$$A\mathbf{v} = \lambda\mathbf{v} \tag{2.55}$$

Here A is some matrix, $\mathbf{v}$ is an eigenvector of that matrix, and λ is the associated eigenvalue. We see that the act of multiplying the matrix A by the eigenvector $\mathbf{v}$ is equivalent to scaling the vector $\mathbf{v}$ by some scalar λ.

It may not be immediately clear how phase kickback is useful, but we will see later on how we can take advantage of this concept to create some powerful algorithms.

2.1.11 Grover's Algorithm

In this section, I describe how Grover's quantum search algorithm works. The purpose of Grover's search algorithm is to search an unsorted database for a specific item or items. There are two key pieces of Grover's algorithm; the Oracle, and the Amplitude Amplification. These pieces work together to return a solution in a time that is exponentially faster than classically possible.

Grover's algorithm is an excellent quantum algorithm for study because it is so general. The general algorithm described is to search an unsorted database for a specific element, and return that element. Let's assume we have a database with N elements. If we used a classical algorithm, we have to individually check each element to see if it is the one we are looking for. In the worst case, the element we are looking for is the last one we checked, so we checked N elements. This gives us the time complexity for this algorithm as $O(N)$. However Grover's algorithm can always return to us the element we are looking for after only $\sqrt{N}$ steps, which gives us the complexity of $O(\sqrt{N})$. Since the algorithm itself is very general, we can obtain this quadratic speed up on a wide variety of problems.

The Oracle is the crux of Grover's algorithm. The rest of the algorithm is completely problem independent. The Oracle is the one piece of the algorithm, and the corresponding circuit, that needs to be designed for a specific problem. Each Grover Oracle is unique to the problem it solves. The purpose of the Oracle is to tag the solution state (item we are looking for) with a minus sign. Physically, this minus sign corresponds to a phase shift of π radians.

In general, the Grover Oracle is "looking" for a solution state, $|\beta\rangle$. The Oracle can tag more than one solution, but for the current discussion, we will assume that the Oracle only tags a single state. The Oracle is a quantum gate that does the following, for any state $|x\rangle$ in our computational basis:

$$O\,|x\rangle = \begin{cases} -\,|x\rangle & \text{if } x = \beta \\ \\ |x\rangle & \text{if } x \neq \beta \end{cases} \tag{2.56}$$

An example will help to illustrate this. In our example, we use two qubits as our "database", with the entire "database" consisting of all the possible basis states of these qubits. These four states are $|00\rangle, |01\rangle, |10\rangle, |11\rangle$. For our example, we will be looking for the element $|01\rangle$.

This Oracle, O takes the mathematical form of a diagonal matrix, where the entry corresponding to our desired state is marked with a negative phase. For our example, the Oracle would be as follows:

$$O = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{array}{l} \\ \leftarrow \beta = 01 \\ \\ \\ \end{array} \tag{2.57}$$

With this representation of our Oracle O, we can treat the Oracle as a gate acting on our quantum state. For our example, O is a gate acting on two qubits. Before applying this Oracle to our quantum state, we will set up our state $|\psi\rangle$ in the equal superposition state

$$|\psi_0\rangle = \frac{1}{2}(|00\rangle + |01\rangle + |10\rangle + |11\rangle) \tag{2.58}$$

In practice, the vast majority of quantum computers start with all their qubits in the $|0\rangle$ state. If our initial state is $|\psi\rangle = |00\rangle$, we can get to the superposition state by applying the Hadamard gate to each qubit. With our state in equal superposition, we can apply our Oracle O

$$|\psi_1\rangle = O\,|\psi_0\rangle = \frac{1}{2}\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & -1 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}\begin{bmatrix}1\\1\\1\\1\end{bmatrix} = \frac{1}{2}\begin{bmatrix}1\\-1\\1\\1\end{bmatrix} = \frac{1}{2}(|00\rangle - |01\rangle + |10\rangle + |11\rangle) \tag{2.59}$$

Now that we have a state where our desired solution state $|\beta\rangle = |01\rangle$ is tagged with a minus sign, we can move onto the next part of Grover's algorithm; amplitude amplification.

At this point, if we measure our state $|\psi_1\rangle$, we are no more likely to get our solution state than we are any other state. This is because the probability of our wave function $|\psi_1\rangle$ collapsing into any specific state, is the absolute value of the amplitude of that state, squared. Since all we have done is flip the phase of our solution state, the absolute value of the amplitude is still the same. This is where the amplitude amplification comes in. The basic idea of amplitude amplification is to increase the amplitude of the "tagged" states, while proportionately decreasing the amplitude of the "un-tagged" states. By doing this, we set up our state $|\psi\rangle$ such that measuring will yield the solution state with very high probability.

The amplification operation is often referred to as the "Diffuser," and it is how I will refer to it throughout this paper. The Diffuser can be thought of as another quantum gate, described generally by:

$$U_s = 2\,|\psi\rangle\,\langle\psi| - I \tag{2.60}$$

where I is the identity matrix. In Grover's original paper, the $|\psi\rangle$ in equation 2.60 was the equal superposition state $|+\rangle^{\otimes n}$ [9]. It was later shown that quantum search is also possible by letting $|\psi\rangle$ be some other state [15], however for the purposes of this paper, Grover's original Diffuser is used. This Diffuser is described as:

$$U_s = 2\left|+\right\rangle^{\otimes n}\left\langle +\right|^{\otimes n} - I \tag{2.61}$$

The effect of this gate, is to amplify any state with a phase of $e^{i\pi}$ while diminishing all other states. This is difficult to see just by looking at the gate, so we will calculate U_s for the two qubit case, and apply it to our example state $|\psi_1\rangle$. With two qubits, the resulting U_s gate will be a 4×4 matrix.

$$\begin{aligned}
U_s &= 2\left|+\right\rangle^{\otimes n}\left\langle +\right|^{\otimes n} - I \\
&= 2\left(\frac{1}{2}\begin{bmatrix}1\\1\\1\\1\end{bmatrix} \cdot \frac{1}{2}\begin{bmatrix}1 & 1 & 1 & 1\end{bmatrix}\right) - \begin{bmatrix}1 & 0 & 0 & 0\\0 & 1 & 0 & 0\\0 & 0 & 1 & 0\\0 & 0 & 0 & 1\end{bmatrix} \\
&= 2\left(\frac{1}{4}\begin{bmatrix}1 & 1 & 1 & 1\\1 & 1 & 1 & 1\\1 & 1 & 1 & 1\\1 & 1 & 1 & 1\end{bmatrix}\right) - \begin{bmatrix}1 & 0 & 0 & 0\\0 & 1 & 0 & 0\\0 & 0 & 1 & 0\\0 & 0 & 0 & 1\end{bmatrix} \\
&= \frac{1}{2}\begin{bmatrix}-1 & 1 & 1 & 1\\1 & -1 & 1 & 1\\1 & 1 & -1 & 1\\1 & 1 & 1 & -1\end{bmatrix}
\end{aligned} \tag{2.62}$$

We can now apply the U_s gate to our state $|\psi_1\rangle$:

$$|\psi_2\rangle = U_s |\psi_1\rangle = \frac{1}{2}\begin{bmatrix} -1 & 1 & 1 & 1 \\ 1 & -1 & 1 & 1 \\ 1 & 1 & -1 & 1 \\ 1 & 1 & 1 & -1 \end{bmatrix} \frac{1}{2}\begin{bmatrix} 1 \\ -1 \\ 1 \\ 1 \end{bmatrix} = \frac{1}{4}\begin{bmatrix} 0 \\ 4 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \tag{2.63}$$

Incredibly, we see that after applying U_s to the state $|\psi_1\rangle$, we end up with a state $|\psi_2\rangle = |01\rangle$. If we now measure the state $|\psi_2\rangle$ we will, with 100% certainty, obtain our desired state of $|01\rangle$! The effect we see here, of bringing the desired states' amplitude to 1 while reducing all others to 0, is a special case of Grover's algorithm when the number of solutions is equal to one quarter the number of all possible states.

The overall effect of the algorithm is difficult to see mathematically, but thankfully there is a very useful geometric interpretation. Initially, our state $|\psi\rangle$ is prepared in the equal superposition, and is described by $|\psi\rangle = |+\rangle^{\otimes n}$, where n is the number of qubits. However, we can also describe $|\psi\rangle$ as the superposition of all *solution* states, and *non-solution* states. We will let $|\beta\rangle$ represent all of the solution states, and $|\alpha\rangle$ represent all the non-solution states. We will let N represent the total number of basis states in our system. $N = 2^n$ where n is the number of qubits. Additionally, we let M represent the total number of *solutions* to the search problem. This implies that $N - M$ represents the number of *non-solutions* to the problem. We define $\sum_x'$ as the sum over every x which is a solution to the search problem, and $\sum_x''$ as the sum over every x which is not a solution to the search problem. We can then define the normalized states $|\alpha\rangle$ and $|\beta\rangle$ as:

$$|\alpha\rangle \equiv \frac{1}{\sqrt{N-M}} \sum_x^{\prime\prime} |x\rangle \tag{2.64}$$

$$|\beta\rangle \equiv \frac{1}{\sqrt{M}} \sum_x^{\prime} |x\rangle \tag{2.65}$$

Since $|\psi\rangle$ is the equal superposition of all states, we can express it as

$$|\psi\rangle = \frac{1}{\sqrt{N}} \sum_x^{\prime\prime\prime} |x\rangle \tag{2.66}$$

where $\sum_x^{\prime\prime\prime}$ is the sum over all states $|x\rangle$. Combining these equations, we get

$$\begin{align} |\psi\rangle &= \frac{1}{\sqrt{N}} \sum_x^{\prime\prime\prime} |x\rangle \tag{2.67}\\ &= \frac{1}{\sqrt{N}} \left(\sum_x^{\prime} |x\rangle + \sum_x^{\prime\prime} |x\rangle \right) \tag{2.68}\\ &= \frac{1}{\sqrt{N}} \left(\sqrt{M}\,|\beta\rangle + \sqrt{N-M}\,|\alpha\rangle \right) \tag{2.69}\\ |\psi\rangle &= \sqrt{\frac{N-M}{N}}\,|\alpha\rangle + \sqrt{\frac{M}{N}}\,|\beta\rangle \tag{2.70} \end{align}$$

Our initial state $|\psi\rangle$ is spanned by the space created by $|\alpha\rangle$ and $|\beta\rangle$. We can understand Grover's algorithm by interpreting it as the application of two *reflections* in the space created by $|\alpha\rangle$ and $|\beta\rangle$. The first reflection is performed by our Oracle, and it reflects our state about the vector $|\alpha\rangle$. This is shown in equation 2.71.

$$O(a\,|\alpha\rangle + b\,|\beta\rangle) = a\,|\alpha\rangle - b\,|\beta\rangle \tag{2.71}$$

The second reflection is performed by the Diffuser, described by $U_s = 2\,|+\rangle^{\otimes n}\,\langle+|^{\otimes n} - I$. Applying this to our state performs a reflection about the vector $|+\rangle^{\otimes n}$, which represents the equal superposition state. Since the product of two reflections is a

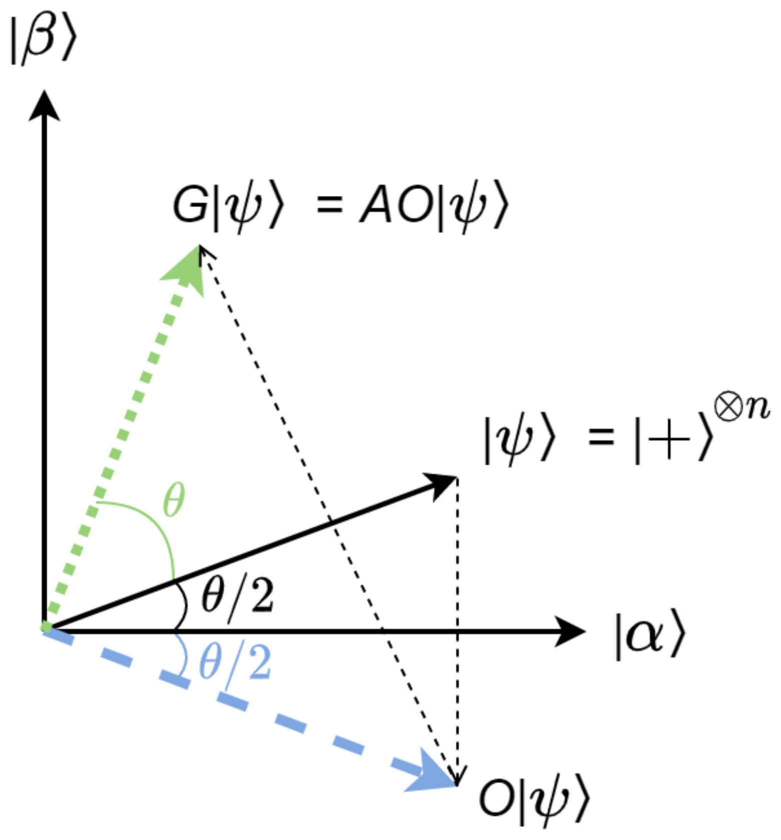


Figure 2.20: Visualization of Grover Iteration. Thick blue dashed vector is result of applying just the Oracle. Thick green dotted vector is result of applying Diffuser after the Oracle. Thin dashed lines indicate transitions.

rotation, we can think of the Grover operator as a rotation in the space spanned by $|\alpha\rangle$ and $|\beta\rangle$. This is shown in figure 2.20. We define $\theta/2$ as the angle between our inital state $|\psi\rangle$, and the $|\alpha\rangle$ axis (which is all the non-solutions). We see that the application of the Oracle reflects $|\psi\rangle$ about the non-solution axis. Then we apply the Diffuser, which reflects about the equal superposition state $|+\rangle^{\otimes n}$. We can see that the overall effect of the Grover operator $G = (2\,|+\rangle^{\otimes n}\,\langle +|^{\otimes n} - I)O$, is to rotate our state vector by θ in a counter-clockwise direction. Additionally, we can see that each subsequent application of the Grover operator G rotates our state vector by another θ radians.

From this geometric visualization, we can see that the purpose of Grover's algorithm is to apply the Grover operation the correct number of times so as to rotate our state vector to be as close to $|\beta\rangle$ as possible. If we take θ to be in radians, then the ideal number of rotations can be found by

$$R = CI\left(\frac{\arccos\left(\sqrt{M/N}\right)}{\theta}\right) \tag{2.72}$$

where CI returns the integer closest to the input [16].

This is great, however it does present a problem; in order to know how many times to apply the Grover operator, we need to know the angle θ between the initial superposition state $|+\rangle^{\otimes n}$ and the non-solution axis $|\alpha\rangle$, *and* the number of solutions M. The value N is the size of the search space and is $N = 2^n$ where n is the number of qubits used. Thankfully, there is an algorithm that we can use called Quantum Counting, which finds θ for a specific Grover Oracle. Using this θ, we can estimate the number of solutions that a specific search Oracle has. To do this, we return to the geometric interpretation of Grover's algorithm. Figure 2.21 shows the initial state with $|\alpha\rangle$ and $|\beta\rangle$ components labeled. We recall that the state $|+\rangle^{\otimes n}$ represents the equal superposition state, which contains all the solutions and non-solutions to our problem. We also recall that $|\alpha\rangle$ represents all the non-solutions, while $|\beta\rangle$ represents all the solutions to our problem. The superposition state can be expressed as

$$|+\rangle^{\otimes n} = \frac{\sqrt{N-M}}{\sqrt{N}}|\alpha\rangle + \sqrt{\frac{M}{N}}|\beta\rangle \tag{2.73}$$

Where M is the number of solutions, and N is the total number of states. We can also express this state in terms of the angle $\theta/2$

$$|+\rangle^{\otimes n} = \cos\frac{\theta}{2}|\alpha\rangle + \sin\frac{\theta}{2}|\beta\rangle \tag{2.74}$$

Given the following equality

$$\sin\frac{\theta}{2} = \sqrt{\frac{M}{N}} \tag{2.75}$$

we can calculate the number of solutions, M, as

$$M = N \times \left(\sin\frac{\theta}{2}\right)^2 \tag{2.76}$$

Using these algorithms in tandem, we can solve a wide range of problems faster than is possible with classical computers. It turns out that the number of iterations will scale according to $O(\sqrt{\frac{N}{M}})$ [16].

In order to see how we can apply Grover's algorithm to a general problem, we will describe our Oracle O in terms of a function f. We define f as follows:

$$f(x) = \begin{cases} 1 & \text{if } x \text{ is a valid solution} \\ 0 & \text{if } x \text{ is not a solution} \end{cases} \tag{2.77}$$

This function is the key piece of our Oracle that needs to be constructed in the form of a quantum circuit. With this function f, we can redefine the Oracle as follows:

$$O\,|x\rangle = (-1)^{f(x)}\,|x\rangle \tag{2.78}$$

The matrix representation of O is now:

$$O = \begin{bmatrix} (-1)^{f(0)} & 0 & \cdots & 0 \\ 0 & (-1)^{f(1)} & \cdots & 0 \\ \vdots & 0 & \ddots & \vdots \\ 0 & 0 & \cdots & (-1)^{f(2^n-1)} \end{bmatrix} \tag{2.79}$$

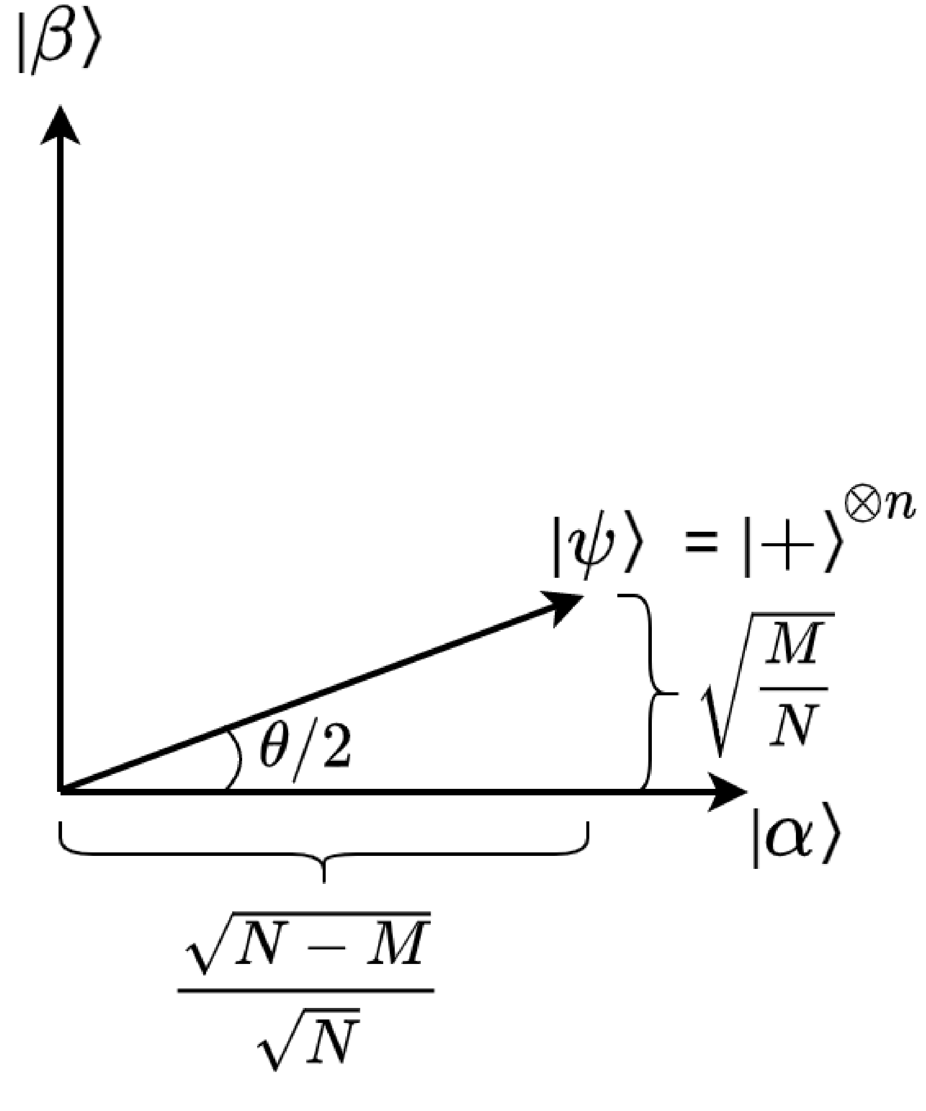

$|\beta\rangle$
$|\psi\rangle = |+\rangle^{\otimes n}$
$\sqrt{\frac{M}{N}}$
$\theta/2$
$|\alpha\rangle$
$\frac{\sqrt{N-M}}{\sqrt{N}}$

Figure 2.21: Geometric visualization of Quantum Search with $|\alpha\rangle$ and $|\beta\rangle$ components of initial state $|+\rangle^{\otimes n}$ explicitly labeled

The key challenge now, is to define that function f, and to create a circuit that implements it.

2.1.12 Using Grover's Algorithm

In this section, I outline the most common way that Grover's algorithm is applied to a general problem. This approach, is to implement an Oracle of the form 2.79 and write the output of the function f to an ancilla qubit. Then this ancilla qubit is used to apply the phase shift (minus sign) by leveraging phase kickback. An ancilla qubit is a qubit that does not encode any piece of the solution, and is just used as a sort of intermediate storage. Ancilla qubits are also often called workspace qubits.

If we return to our example of finding the state $|01\rangle$, we want our function $f(x)$ to return 1 if $x = |01\rangle$, and 0 in all other cases. Thus our desired state is

$$|\psi\rangle = \frac{1}{2}(|00\rangle|0\rangle + |01\rangle|1\rangle + |10\rangle|0\rangle + |11\rangle|0\rangle) = \frac{1}{2}\sum_{x=0}^{3}|x\rangle|f(x)\rangle \tag{2.80}$$

The circuit implementing this function is shown in figure 2.22. Here we see that we first apply an X-gate to qubit 0. For our target state $|01\rangle$, this will turn it into the state $|11\rangle$. Then we apply a Toffoli gate with qubits 0 and 1 as the control, and qubit 2 as the target. Here qubit 2 is our ancilla qubit and qubits 0 and 1 are the data qubits. Finally, we apply another X-gate to qubit 0 to return it to the original state of $|01\rangle$. In this circuit, the ancilla qubit holds the result of $f(x)$. In the full implementation of Grover's algorithm, we initially prepare the data qubits in the equal superposition state, then apply the Oracle. We also initialize the ancilla qubit in the state $|0\rangle$. Thus our initial state before applying the Oracle is $|\psi_0\rangle = \frac{1}{2}(|000\rangle + |010\rangle + |100\rangle + |110\rangle)$.

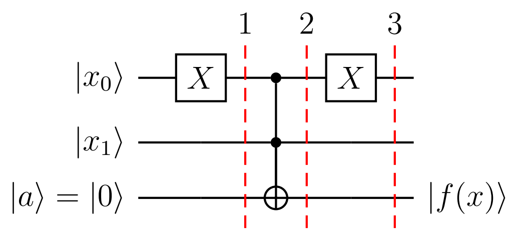


Figure 2.22: Circuit that sets $|a\rangle$ to $|1\rangle$ if $|x_0x_1\rangle = |01\rangle$

In order to make sure that circuit 2.22 actually implements $f(x)$, we will follow the state $|\psi_0\rangle$ as it moves through the circuit:

1. X-gate is applied to qubit 0:

$$|\psi_1\rangle = X_0 |\psi_0\rangle = \frac{1}{2}(X_0 |000\rangle + X_0 |010\rangle + X_0 |100\rangle + X_0 |110\rangle) \tag{2.81}$$

$$= \frac{1}{2}(|100\rangle + |110\rangle + |000\rangle + |010\rangle) \tag{2.82}$$

2. Apply Toffoli gate with qubits 0 and 1 as control, and qubit 2 (ancilla qubit) as target:

$$|\psi_2\rangle = TOF_{(0,1),2} |\psi_1\rangle \tag{2.83}$$

$$= \frac{1}{2}(TOF_{(0,1),2} |100\rangle + TOF_{(0,1),2} |110\rangle + TOF_{(0,1),2} |000\rangle + TOF_{(0,1),2} |010\rangle) \tag{2.84}$$

$$= \frac{1}{2}(|100\rangle + |111\rangle + |000\rangle + |010\rangle) \tag{2.85}$$

Here $TOF_{(0,1),2}$ denotes the Toffoli gate with qubits 0 and 1 as control, and qubit 2 as target.

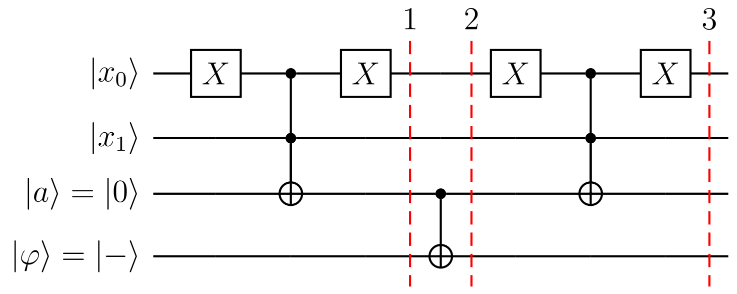


Figure 2.23: Full Oracle for tagging state $|01\rangle$

3. Apply X-gate to qubit 0:

$$|\psi_3\rangle = X_0\,|\psi_2\rangle = \frac{1}{2}(X_0\,|100\rangle + X_0\,|111\rangle + X_0\,|000\rangle + X_0\,|010\rangle) \tag{2.86}$$

$$= \frac{1}{2}(|000\rangle + |011\rangle + |100\rangle + |110\rangle) \tag{2.87}$$

$$= \frac{1}{2}\sum_{x=0}^{3}|x\rangle\,|f(x)\rangle \tag{2.88}$$

This is great! We now have a circuit that creates our desired state $\frac{1}{2}\sum_{x=0}^{3}|x\rangle\,|f(x)\rangle$. However, in order for this to be an Oracle, we need to apply a minus sign to our solution state. In order to do this, we take advantage of phase kickback. We will create another qubit, called the output qubit, and prepare it in the state $|-\rangle$. Then after we apply our function $f(x)$, add a cNOT gate with our ancilla qubit as the control, and the output qubit as the target. This circuit is shown in figure 2.23. The exact form of this circuit may be confusing at first, but after stepping through it, the form will become clear. As before, we initialize both data qubits in a superposition, and the ancilla qubit in $|0\rangle$. We also initialize the output qubit in the state $|-\rangle$. Thus our initial state $|\psi_0\rangle = \frac{1}{2}(|000-\rangle + |010-\rangle + |100-\rangle + |110-\rangle)$, with the form $|\psi_0\rangle = |x_0x_1a\varphi\rangle$, where x_0, x_1 are the data qubits, a is the ancilla, and φ is the output qubit. With this as the initial state, the circuit 2.23 does the following:

1. Apply the circuit implementing $f(x)$. Note that since φ is untouched, it will stay the same throughout:

$$|\psi_1\rangle = F\,|\psi_0\rangle = \frac{1}{2}(|000-\rangle + |011-\rangle + |100-\rangle + |110-\rangle) \tag{2.89}$$

   Here F represents the circuit in figure 2.22

2. Apply the cNOT gate with a as the control, and φ as the target:

$$\begin{aligned}
|\psi_2\rangle &= cNOT_{a,\varphi}\,|\psi_1\rangle && (2.90)\\
&= \frac{1}{2}(cNOT_{a,\varphi}\,|000-\rangle + cNOT_{a,\varphi}\,|011-\rangle + cNOT_{a,\varphi}\,|100-\rangle \\
&\quad + cNOT_{a,\varphi}\,|110-\rangle) && (2.91)\\
&= \frac{1}{2}(|000-\rangle + cNOT_{a,\varphi}\,|011-\rangle + |100-\rangle + |110-\rangle) && (2.92)\\
&= \frac{1}{2}\left(|000-\rangle + |011\rangle \otimes X\left(\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)\right) + |100-\rangle + |110-\rangle\right) && (2.93)\\
&= \frac{1}{2}\left(|000-\rangle + |011\rangle \otimes \left(\frac{1}{\sqrt{2}}(|1\rangle - |0\rangle)\right) + |100-\rangle + |110-\rangle\right) && (2.94)\\
&= \frac{1}{2}\left(|000-\rangle + |011\rangle \otimes \left(-\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)\right) + |100-\rangle + |110-\rangle\right) && (2.95)\\
&= \frac{1}{2}(|000-\rangle - |011-\rangle + |100-\rangle + |110-\rangle) && (2.96)
\end{aligned}$$

   Here $cNOT_{a,\varphi}$ represents a cNOT gate with a as the control, and φ as the target

3. Re-apply the circuit 2.22. This step is called un-computation, and returns a to its original state

$$|\psi_3\rangle = F\,|\psi_2\rangle = \frac{1}{2}(|000-\rangle - |010-\rangle + |100-\rangle + |110-\rangle) \tag{2.97}$$

We now have a fully functional Oracle! Once this has been created, we apply the Diffuser to the data qubits in order to amplify the marked states' amplitudes.

We recall that the Diffuser is defined by:

$$U_s = 2 \left|+\right\rangle^{\otimes n} \left\langle+\right|^{\otimes n} - I \tag{2.98}$$

In order to implement this operation in a quantum circuit, we perform some conversions. For the first conversion, we recall that the geometric significance of the Diffuser U_s was to *reflect* the state we apply it to about the equal superposition state $\left|+\right\rangle^{\otimes n}$. This reflection consists of adding a minus sign to every state orthogonal to $\left|+\right\rangle^{\otimes n}$. Since this is difficult to explicitly implement, what we do instead is convert the $\left|+\right\rangle^{\otimes n}$ state to the $\left|00\ldots0\right\rangle$ state, reflect about the $\left|00\ldots0\right\rangle$ state, then convert back. Thankfully, we already know how to transform the $\left|+\right\rangle^{\otimes n}$ state to $\left|00\ldots0\right\rangle$; we apply the Hadamard gate to each qubit.

$$H^{\otimes n} \left|+\right\rangle^{\otimes n} = \left|00\ldots0\right\rangle \tag{2.99}$$

In order to reflect about the $\left|00\ldots0\right\rangle$ state, we add a minus sign to every state orthogonal to $\left|00\ldots0\right\rangle$. This new reflection U_0 is defined as:

$$U_0 = \begin{bmatrix} 1 & 0 & \ldots & 0 \\ 0 & -1 & \ldots & 0 \\ \vdots & 0 & \ddots & \vdots \\ 0 & 0 & \ldots & -1 \end{bmatrix} \tag{2.100}$$

We can now define U_s as

$$U_s = H^{\otimes n} U_0 H^{\otimes n} \tag{2.101}$$

The second trick we do, has to do with the implementation of U_0. What we can do, is convert the state $\left|00\ldots0\right\rangle$ to the state $\left|11\ldots1\right\rangle$, apply a minus sign to state $\left|11\ldots1\right\rangle$, then convert back $|11\dots1\rangle \rightarrow |00\dots0\rangle$. This will give us the following matrix

$$\begin{bmatrix} -1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & 0 & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix} \tag{2.102}$$

We can then factor out a global phase of (-1), and this will give us the gate U_0. In order to perform the transformation $|00\dots0\rangle \rightarrow |11\dots1\rangle$, we apply the X-gate to each qubit:

$$X^{\otimes n}|00\dots0\rangle = |11\dots1\rangle \tag{2.103}$$

Finally, in order to apply a minus sign to only the $|11\dots1\rangle$ state, we use a multi-controlled Z gate (MCZ). Thus we can see that

$$U_0 = -X^{\otimes n} MCZ X^{\otimes n} \tag{2.104}$$

and putting this all together, we get

$$U_s = -H^{\otimes n} X^{\otimes n} MCZ X^{\otimes n} H^{\otimes n} \tag{2.105}$$

When using this Diffuser for Quantum Search, we can ignore the phase (-1) since it is global. However, when using this Diffuser in the Quantum Counting algorithm, we cannot ignore this phase. We will see how to deal with this phase at the end of section 2.1.15. The general circuit that implements this Diffuser for any n qubits is shown in figure 2.24.

Returning to our example of finding the state $|01\rangle$, we add the two qubit Diffuser to our circuit 2.23, giving us the circuit 2.25. This circuit depicts the entire Grover

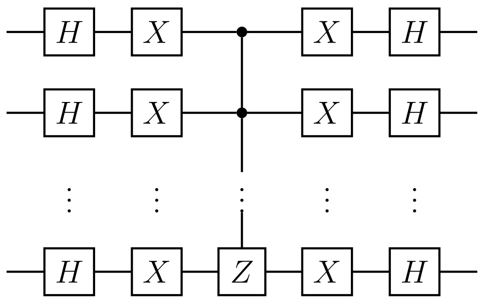


Figure 2.24: General Diffuser circuit

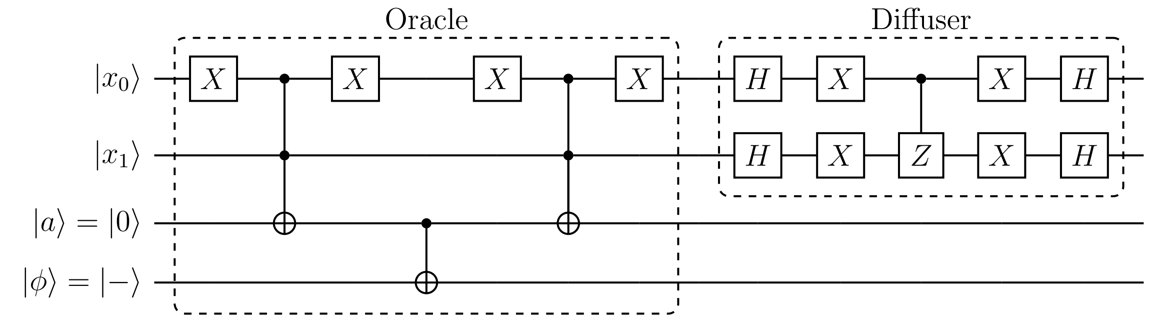


Figure 2.25: Full Grover operator for finding state $|01\rangle$

operator for this specific problem. Measuring qubits $|x_0x_1\rangle$ after the Diffuser will yield us our solution of $|01\rangle$.

2.1.13 Quantum Fourier Transform

Before we can dive into how the Quantum Counting algorithm works, we must first discuss two cornerstone quantum algorithms; Quantum Fourier Transform, and Quantum Phase Estimation. The Quantum Phase Estimation (QPE) algorithm uses the Quantum Fourier Transform (QFT), so we will start with the QFT.

Before diving into the Quantum Fourier Transform, it is useful to remind ourselves of the classical Fourier Transform in both continuous, and discrete space. Note that much of the following discussion is adapted from [17]. The continuous Fourier Transform is given by:

$$F(\omega) = \int_{-\infty}^{\infty} f(t)e^{i\omega t}dt \tag{2.106}$$

where $f(t)$ is a continuous time function that will give us an exact value at any point in time t. $F(\omega)$ describes the same function $f(t)$, but in the frequency domain, instead of the time domain. In order to move from the continuous Fourier Transform to the Discrete Fourier Transform (DFT), we can imagine that we have sampled N points from our function $f(t)$ each at distinct time intervals T. We can represent these samples as $f[0], f[1], \ldots f[k], \ldots f[N-1]$. Plugging this into equation 2.106, we observe that our function only exists at the sampled points:

$$\begin{aligned} F(\omega) &= \int_{0}^{T(N-1)} f(t)e^{i\omega t}dt \\ &= f[0]e^{0} + f[1]e^{i\omega T} + \cdots + f[k]e^{i\omega kT} + \cdots + f[N-1]e^{i\omega(N-1)T} \\ &= \sum_{k=0}^{N-1} f[k]e^{i\omega kT} \end{aligned} \tag{2.107}$$

We recall that we can also perform the Fourier Transform over a finite interval (often T_0) if our function $f(t)$ is periodic. Since our discrete function has a finite number of samples, we treat the entire sample set as one period of our function (i.e. $f(0)$ to $f(N-1)$ is the same as $f(N)$ to $f(2N-1)$). This lets us evaluate the transform with the *fundamental* frequency ω_0. Since we are treating our entire sample size as a single period, $\omega_0 = \frac{2\pi}{NT}$ rad. We then see that our ω is then restricted to the following set:

$$\omega = 0, \frac{2\pi}{NT}, \frac{2\pi}{NT} \times 2, \ldots \frac{2\pi}{NT} \times j, \ldots \frac{2\pi}{NT} \times (N-1) \tag{2.108}$$

Plugging this into the sum derived in equation 2.107, we arrive at a form that will be useful for discussion of the Quantum Fourier Transform:

$$F[j] = \sum_{k=0}^{N-1} f[k]e^{i\frac{2\pi j}{NT}kT} = \sum_{k=0}^{N-1} f[k]e^{i\frac{2\pi}{N}jk} \tag{2.109}$$

Returning to the QFT, we recall that we often describe a quantum state by its amplitude vector. Because of this, it is useful to think of the Discrete Fourier Transform as mapping an amplitude vector $(a_0, a_1, \ldots, a_{N-1})$ to the amplitude vector $(b_0, b_1, \ldots, b_{N-1})$. This mapping is then described by:

$$b_j = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} a_k e^{i\frac{2\pi}{N}jk} \tag{2.110}$$

The $\frac{1}{\sqrt{N}}$ is the normalization factor that we split between our definition of the Fourier Transform, and the Inverse Fourier Transform. Splitting it in this way is useful for the QFT.

The Quantum Fourier Transform acts on the state $\sum_{i=0}^{N-1} a_i \left|i\right\rangle$, and maps it to the state $\sum_{i=0}^{N-1} b_i \left|i\right\rangle$, in accordance with equation 2.110. In this form, we see that the QFT only acts on the amplitudes themselves, and leaves the states untouched. Another interpretation of the QFT, is to think of it acting on the states themselves, and leaving the amplitudes untouched. Thinking about it in this fashion, we would express the QFT as the following map:

$$\left|x\right\rangle \mapsto \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} e^{i\frac{2\pi}{N}xy} \left|y\right\rangle \tag{2.111}$$

We notice that in this form, the x and y in the $|\ \rangle$'s are the same x and y that are in the exponent. For example, assume we are working with four qubits, $N = 2^4 = 16$, and are mapping the state $\left|x\right\rangle = \left|1000\right\rangle$. The state $\left|x\right\rangle$ is written in binary, but it may also be written in decimal form, $\left|1000\right\rangle = \left|8\right\rangle$. For the entire sum, the value of x will be 8. The value y in the sum will go from 0 to 15, with $\left|0\right\rangle = \left|0000\right\rangle$ and $\left|15\right\rangle = \left|1111\right\rangle$. We may also express the QFT as a unitary matrix, which is very useful since unitary matrices are how we represent quantum gates:

$$U_{QFT} = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} e^{i\frac{2\pi}{N}xy} \left|y\right\rangle \left\langle x\right| \tag{2.112}$$

Calculating U_{QFT} will give us a quantum gate that we can apply to any state, which will have the effect of applying the QFT to that state.

Now that we have a formula for computing the QFT, we can explore the intuition behind the QFT, and why it is a useful action to be able to perform. The QFT is an operation that transforms our state between two different *bases.* Usually, when talking about qubits for computation, we express those qubits in the Z-basis, which consists of the states $|0\rangle$ and $|1\rangle$. The QFT will transform a state from the Z-basis, to a state in the Fourier basis. We have actually already seen the gate that performs this action on a single qubit; the Hadamard gate. The H-gate is the single qubit QFT, and transforms Z-basis states $|0\rangle$ and $|1\rangle$ to Fourier basis states $|+\rangle$ and $|-\rangle$. We can derive the H-gate from equation 2.112 by letting $N = 2^1 = 2$, i.e. the single qubit U_{QFT}:

$$U_{QFT} = \frac{1}{\sqrt{2}} \sum_{x=0}^{1} \sum_{y=0}^{1} e^{i\frac{2\pi}{2}xy} |y\rangle \langle x| \tag{2.113}$$

$$= \frac{1}{\sqrt{2}} \left(e^{i2\pi\frac{0\cdot 0}{2}} |0\rangle \langle 0| + e^{i2\pi\frac{0\cdot 1}{2}} |1\rangle \langle 0| + e^{i2\pi\frac{1\cdot 0}{2}} |0\rangle \langle 1| + e^{i2\pi\frac{1\cdot 1}{2}} |1\rangle \langle 1| \right) \tag{2.114}$$

$$= \frac{1}{\sqrt{2}} \left(e^{0} |0\rangle \langle 0| + e^{0} |1\rangle \langle 0| + e^{0} |0\rangle \langle 1| + e^{i\pi} |1\rangle \langle 1| \right) \tag{2.115}$$

$$= \frac{1}{\sqrt{2}} \left(\begin{bmatrix} 1 \\ 0 \end{bmatrix} \begin{bmatrix} 1 & 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} \begin{bmatrix} 0 & 1 \end{bmatrix} - \begin{bmatrix} 0 \\ 1 \end{bmatrix} \begin{bmatrix} 0 & 1 \end{bmatrix} \right) \tag{2.116}$$

$$= \frac{1}{\sqrt{2}} \left(\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} - \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \right) \tag{2.117}$$

$$= \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{2.118}$$

As we increase the number of qubits in our state, each Z-basis state will have a representation in the Fourier basis, and the QFT is how we get there.

A useful example to help see what is happening, is counting. When counting in the Z-basis, we can apply the same logic for counting with classical bits, to counting with qubits. For example, with three qubits, we can represent the numbers 0-7. We let the state $|000\rangle$ represent the number 0, $|001\rangle$ represents the number 1, and so on. Finally we let $|111\rangle$ represent the number 7. In this way, we have counted from 0 to 7 in the Z-basis. We can also observe on a Bloch Sphere how the individual qubits changed as we counted. Figure 2.26 shows the Bloch Sphere representations for the numbers 0, through 7. We can see that the least significant qubit, q_2, flips every time we move to the next number. The next qubit, q_1, flips every other number. Finally, q_0 flips every four numbers. If we added another qubit and counted to 15, we would see that this new qubit flips every 8 numbers. Continuing this pattern, we observe that the

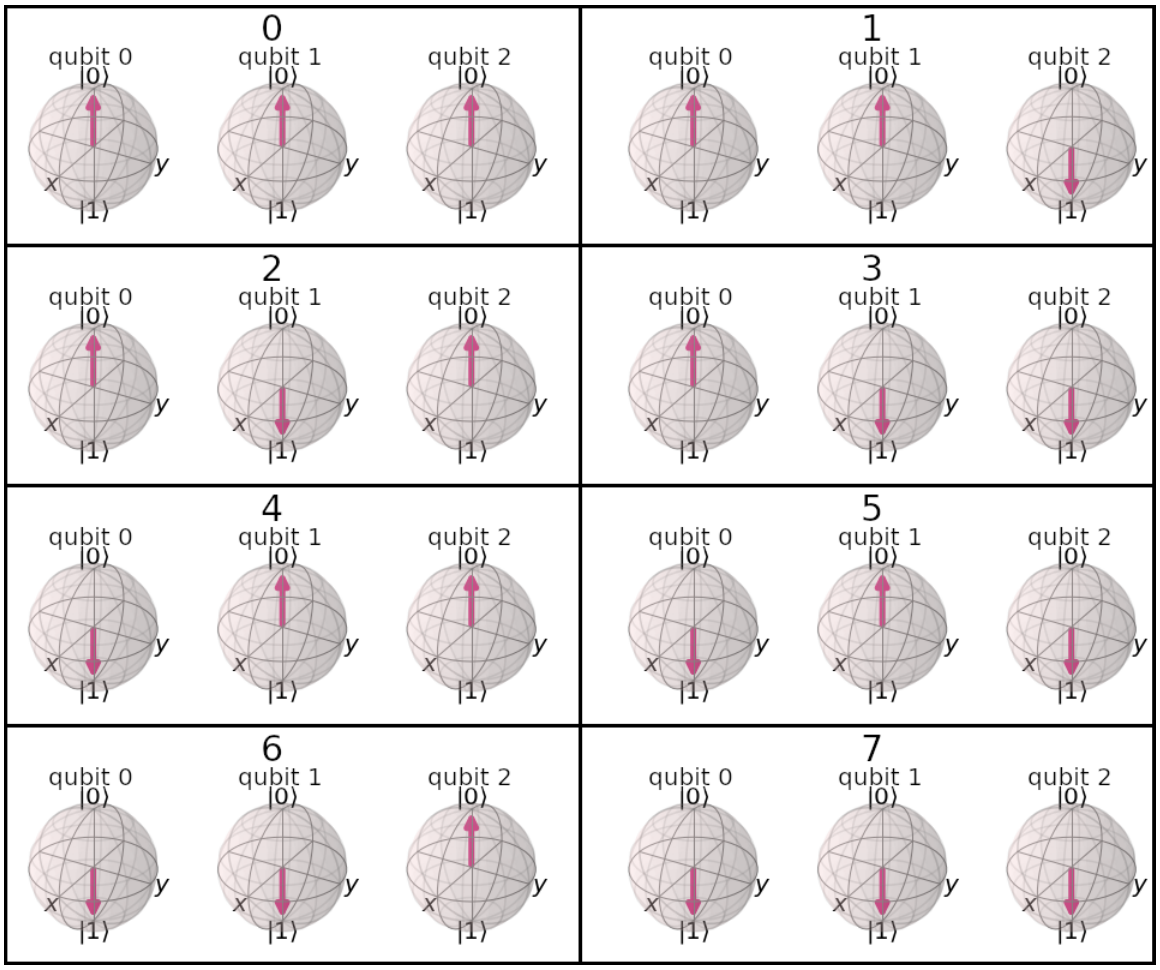


Figure 2.26: Three qubit representations of numbers 0 through 7 in the Z-basis on the Bloch Sphere

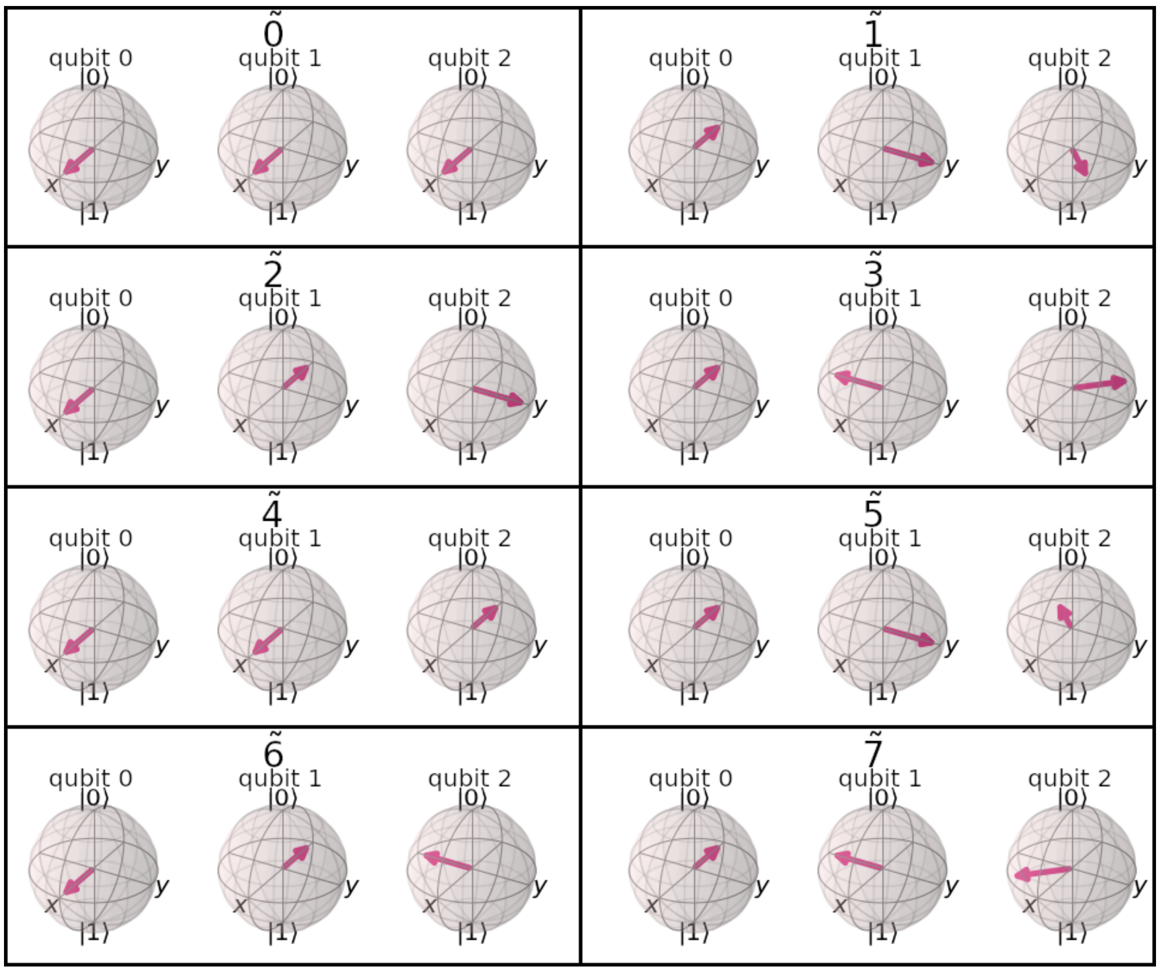


Figure 2.27: Three qubit representations of numbers 0 through 7 in the Fourier-basis on the Bloch Sphere

qubits flip every 2^{n-1} numbers, where n is the number of qubits. We also notice that the qubits simply flip between the two poles of the Bloch Sphere (i.e. states $|0\rangle$ and $|1\rangle$).

When we count in the Fourier basis, the picture changes. Figure 2.27 shows how we would represent the same numbers, 0-7, in the Fourier basis. There are two stark differences to representing numbers in the Fourier basis as opposed to the Z-basis. The first is that we represent the numbers as rotations around the z-axis. The second is that every qubit rotates for every number, and the angle of rotations changes per qubit. Notice that in this picture, q_2 makes one full rotation around the Bloch Sphere when counting from 0-7. q_1 makes two full rotations around the Bloch Sphere, and q_0 makes four full rotations around the Bloch Sphere. Essentially, the angle of rotation doubles with each successive qubit, as we count. For q_2, the angle is $\frac{x}{2^n} \times 2\pi$ radians, where x is the number we are trying to store, and n is the number of qubits in our system. If we are trying to store the number 3, the angle for q_2 is $\frac{3}{8} \times 2\pi$ radians, the angle for q_1 is double that, at $\frac{6}{8} \times 2\pi$ radians, and finally the angle for q_0 is double q_1's angle; $\frac{12}{8} \times 2\pi$ radians.

Equation 2.111 gives us a useful representation of the QFT as it acts on the qubit states. However, it is not clear from this form how we could actually *implement* the transform with a quantum circuit. In order to assist with the implementation, we expand the form of equation 2.111 to describe the action on a general state with $N = 2^n$ basis states. This expansion and the following circuit was adapted from [12]. We denote this as QFT_N. The state that this action is performed on is $|x\rangle = |x_0 x_1 \ldots x_{n-1}\rangle$ where x_0 is the most significant bit.

$$QFT_N |x\rangle = \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} e^{i\frac{2\pi}{N}xy} |y\rangle \tag{2.119}$$

$$= \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} e^{2\pi ixy/2^n} |y\rangle \tag{2.120}$$

$$= \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} e^{2\pi ix(\sum_{k=0}^{n-1} 2^{n-k}y_k)/2^n} |y_0 \dots y_{n-1}\rangle \tag{2.121}$$

$$= \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} e^{2\pi ix(\sum_{k=0}^{n-1} y_k/2^k)} |y_0 \dots y_{n-1}\rangle \tag{2.122}$$

$$= \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} \prod_{k=0}^{n-1} e^{2\pi ixy_k/2^k} |y_0 \dots y_{n-1}\rangle \tag{2.123}$$

$$= \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} e^{\frac{2\pi i}{2}xy_0} \times e^{\frac{2\pi i}{2^2}xy_1} \times \cdots \times e^{\frac{2\pi i}{2^n}xy_{n-1}} |y_0 \dots y_{n-1}\rangle \tag{2.124}$$

$$= \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} e^{\frac{2\pi i}{2}xy_0} |y_0\rangle \otimes e^{\frac{2\pi i}{2^2}xy_1} |y_1\rangle \otimes \cdots \otimes e^{\frac{2\pi i}{2^n}xy_{n-1}} |y_{n-1}\rangle \tag{2.125}$$

$$= \frac{1}{\sqrt{N}} \left(\sum_{y_0=0}^{1} e^{\frac{2\pi i}{2}xy_0} |y_0\rangle \otimes \sum_{y_1=0}^{1} e^{\frac{2\pi i}{2^2}xy_1} |y_1\rangle \otimes \cdots \otimes \sum_{y_{n-1}=0}^{1} e^{\frac{2\pi i}{2^n}xy_{n-1}} |y_{n-1}\rangle \right) \tag{2.126}$$

$$= \frac{1}{\sqrt{N}} \left(|0\rangle + e^{\frac{2\pi i}{2}x} |1\rangle \right) \otimes \left(|0\rangle + e^{\frac{2\pi i}{2^2}x} |1\rangle \right) \otimes \cdots \otimes \left(|0\rangle + e^{\frac{2\pi i}{2^n}x} |1\rangle \right) \tag{2.127}$$

In the above equation, the first line 2.119 is simply a reproduction of equation 2.111. In line 2.120 we substitute $N = 2^n$. In line 2.121 we represent y as a binary number, $y = 2^{n-1}y_0 + 2^{n-2}y_1 + \cdots + 2^1 y_{n-2} + 2^0 y_{n-1}$ where $y_0 \dots y_{n-1}$ is placed in the $|\ \rangle$ and the full binary representation is placed in the exponent. In line 2.122 we bring the $\frac{1}{2^n}$ into the sum; $\frac{2^{n-k}}{2^n} = \frac{1}{2^k}$. In line 2.123 we expand the exponential of a sum to a product of exponentials. In line 2.124 we expand the product of exponentials. In line 2.125 we first split up our state $|y_0 \dots y_{n-1}\rangle$ by taking advantage of the fact that we can represent a multi-qubit state as the tensor products of individual qubits, $|y_0 \dots y_{n-1}\rangle = |y_0\rangle \otimes \dots \otimes |y_{n-1}\rangle$. Then we do some rearranging; grouping terms that pertain to with the same qubit together. In line 2.126 we expand the sum to sum over individual qubits instead of the overall y value, $\sum_{y=0}^{N-1} = \sum_{y_0=0}^{1} \sum_{y_1=0}^{1} \cdots \sum_{y_{n-1}=0}^{1}$. Finally, we arrive at equation 2.127 which expands all the sums. Note that each $|0\rangle$ term actually has a e^0 factor, but since $e^0 = 1$, we leave it out. This form is very useful when it comes to implementing the circuit that performs the QFT.

In order to implement the QFT in the form of 2.127, we actually only need two gates. The first is the Hadamard gate. We have already explored the H-gate, and have seen that it performs the following actions:

$$H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \tag{2.128}$$

$$H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \tag{2.129}$$

Recalling that $e^0 = 1$ and $e^{i\pi} = -1$, we can generalize the action of the H-gate, on a state $|x_k\rangle$:

$$H|x_k\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle + e^{\frac{2\pi i}{2}x_k}|1\rangle\right) \tag{2.130}$$

$$x_k \in [0, 1]$$

We notice that this is very similar to the first tensor of equation 2.127 but instead of x, we have x_k. This difference will be addressed soon.

The next gate that we need is the two-qubit controlled rotation gate $CROT_k$, which is the controlled form of the single-qubit rotation gate $UROT_k$. It is called a "rotation" gate because it applies a relative phase to the qubit state, which essentially "rotates" the qubit around the Bloch Sphere. $UROT_k$ is described by the matrix

$$\begin{bmatrix} 1 & 0 \\ 0 & e^{\frac{2\pi i}{2^k}} \end{bmatrix} \tag{2.131}$$

and $CROT_k$ is described by:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{\frac{2\pi i}{2^k}} \end{bmatrix} \tag{2.132}$$

Here k is a parameter that we pick, which determines how much of a rotation we apply. For the $CROT_k$ gate, the first qubit is our control, and the second qubit is the target. When we apply the $CROT_k$ gate to a general two-qubit case $|x_i x_j\rangle$, we have the following two cases:

$$CROT_k |0x_j\rangle = |0x_j\rangle \tag{2.133}$$

and

$$CROT_k |1x_j\rangle = e^{\frac{2\pi i}{2^k} x_j} |1x_j\rangle \tag{2.134}$$

It is also useful to note how this gate behaves when the target qubit is in superposition $|+\rangle$:

$$CROT_k |+x_j\rangle = CROT_k \left(\frac{1}{\sqrt{2}} (|0\rangle + |1\rangle) \otimes |x_j\rangle \right) = \frac{1}{\sqrt{2}} \left(|0\rangle + e^{\frac{2\pi i}{2^k} x_j} |1\rangle \right) \otimes |x_j\rangle \tag{2.135}$$

The phase $e^{\frac{2\pi i}{2^k} x_j}$ is applied to the first qubit because of phase kickback.

With these two gates, we can create a circuit that will apply the QFT to a general state $|x_0 \ldots x_{n-1}\rangle$. This circuit is shown in figure 2.28.

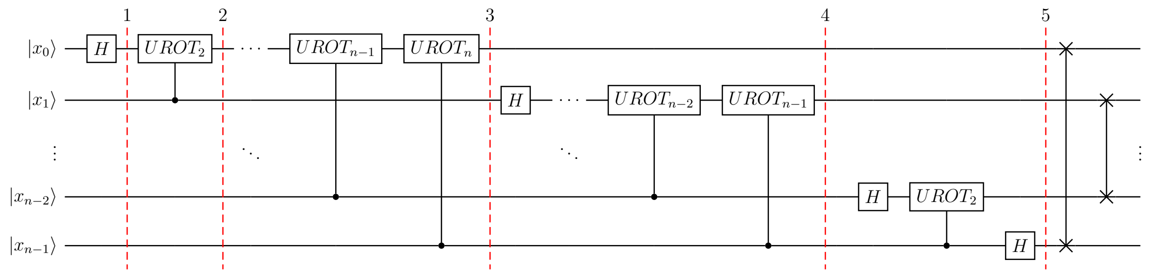


Figure 2.28: General circuit for QFT acting on state $|x_0 \dots x_{n-1}\rangle$

Starting with our initial state $|\psi_0\rangle = |x_0 x_1 \dots x_{n-1}\rangle$, the circuit acts on this state in the following way (note each number in the following list describes the state at the numbered slice in figure 2.28):

1. We start by applying the H-gate to $|x_0\rangle$, which transforms our overall state as follows:

$$|\psi_1\rangle = H_0 |x_0 x_1 \dots x_{n-1}\rangle = \frac{1}{\sqrt{2}} \left[|0\rangle + e^{\frac{2\pi i}{2} x_0} |1\rangle \right] \otimes |x_1 x_2 \dots x_{n-1}\rangle \tag{2.136}$$

2. We then apply the $UROT_2$ gate on $|x_0\rangle$ with $|x_1\rangle$ as the control:

$$|\psi_2\rangle = UROT_2 |\psi_1\rangle = \frac{1}{\sqrt{2}} \left[|0\rangle + e^{\frac{2\pi i}{2} x_0 + \frac{2\pi i}{2^2} x_1} |1\rangle \right] \otimes |x_1 x_2 \dots x_{n-1}\rangle \tag{2.137}$$

3. We then apply $UROT_3$ through $UROT_n$ to $|x_0\rangle$, controlled by qubits $|x_2\rangle$ through $|x_{n-1}\rangle$ respectively. After applying $UROT_n$ to $|x_0\rangle$, controlled by $|x_{n-1}\rangle$, we have the following state:

$$|\psi_3\rangle = \frac{1}{\sqrt{2}} \left[|0\rangle + e^{\frac{2\pi i}{2} x_0 + \frac{2\pi i}{2^2} x_1 + \dots + \frac{2\pi i}{2^{n-1}} x_{n-2} + \frac{2\pi i}{2^n} x_{n-1}} |1\rangle \right] \otimes |x_1 x_2 \dots x_{n-1}\rangle \tag{2.138}$$

In order to simplify this, we first note that:

$$x = 2^{n-1} x_0 + 2^{n-2} x_1 + \dots + 2^1 x_{n-2} + 2^0 x_{n-1} \tag{2.139}$$

Then, we look at the value in the exponent, and factor out $\frac{2\pi i}{2^n}$:

$$\frac{2\pi i}{2}x_0+\cdots+\frac{2\pi i}{2^{n-1}}x_{n-2}+\frac{2\pi i}{2^n}x_{n-1} = \frac{2\pi i}{2^n}(2^{n-1}x_0+\cdots+2^1x_{n-2}+2^0x_{n-1}) \quad (2.140)$$

Combining equations 2.139 and 2.140, we get the following:

$$|\psi_3\rangle = \frac{1}{\sqrt{2}}\left[|0\rangle + e^{\frac{2\pi i}{2^n}x}|1\rangle\right] \otimes |x_1x_2\ldots x_{n-1}\rangle \quad (2.141)$$

4. We then repeat this process on $|x_1\rangle$, with the notable difference that we only go up to $UROT_{n-1}$ (one less gate). This will give us the following state:

$$|\psi_4\rangle = |\tilde{x_0}\rangle \otimes \frac{1}{\sqrt{2}}\left[|0\rangle + e^{\frac{2\pi i}{2}x_1+\frac{2\pi i}{2^2}x_2+\cdots+\frac{2\pi i}{2^{n-3}}x_{n-2}+\frac{2\pi i}{2^{n-1}}x_{n-1}}|1\rangle\right] \otimes |x_2x_3\ldots x_{n-1}\rangle \quad (2.142)$$

where

$$|\tilde{x_0}\rangle = \frac{1}{\sqrt{2}}\left[|0\rangle + e^{\frac{2\pi i}{2^n}x}|1\rangle\right]$$

Performing a similar factorization of the exponent, we get:

$$\frac{2\pi i}{2}x_1+\cdots+\frac{2\pi i}{2^{n-2}}x_{n-2}+\frac{2\pi i}{2^{n-1}}x_{n-1} = \frac{2\pi i}{2^{n-1}}(2^{n-2}x_1+\cdots+2^1x_{n-2}+2^0x_{n-1}) \quad (2.143)$$

At this point, we note that we *cannot* substitute equation 2.139 in, as we have no x_0 in the above equation. In order to address this issue, we add the x_0 term in, then subtract it off (essentially adding 0):

$$\frac{2\pi i}{2^{n-1}}(2^{n-1}x_0+2^{n-2}x_1+2^{n-2}x_2+\cdots+2^1x_{n-2}+2^0x_{n-1}) - \frac{2\pi i}{2^{n-1}}2^{n-1}x_0 \quad (2.144)$$

Which we can re-write as

$$e^{\frac{2\pi i}{2^{n-1}}x} \times e^{-\frac{2\pi i}{2^{n-1}}2^{n-1}x_0} = e^{\frac{2\pi i}{2^{n-1}}x} \times e^{-2\pi i x_0} \quad (2.145)$$

Since $x_0 \in [0,1]$, we observe that the term $e^{-2\pi i x_0}$ is always 1 since $e^0 = e^{-2\pi i} = 1$. Thus we can safely re-write our state as follows:

$$|\psi_4\rangle = \frac{1}{\sqrt{2}}\left[|0\rangle + e^{\frac{2\pi i}{2^n}x}|1\rangle\right] \otimes \frac{1}{\sqrt{2}}\left[|0\rangle + e^{\frac{2\pi i}{2^{n-1}}x}|1\rangle\right] \otimes |x_2 x_3 \ldots x_{n-1}\rangle \quad (2.146)$$

5. After performing the same process on qubits $2 \ldots n-1$, we end up with the state

$$\frac{1}{\sqrt{2}}\left[|0\rangle + e^{\frac{2\pi i}{2^n}x}|1\rangle\right] \otimes \frac{1}{\sqrt{2}}\left[|0\rangle + e^{\frac{2\pi i}{2^{n-1}}x}|1\rangle\right] \otimes \cdots \otimes \frac{1}{\sqrt{2}}\left[|0\rangle + e^{\frac{2\pi i}{2^1}x}|1\rangle\right] \quad (2.147)$$

   This is the same state as describe in equation 2.127, with the exception that the ordering of the qubits is flipped. Thus we utilize the swap gates at the end of our circuit to fully complete our QFT circuit.

We have now derived a circuit that can implement the QFT for any number of qubits! Using this template, we can apply the QFT to any state we want. Additionally, it is worth noting that if we want to apply the *inverse* Quantum Fourier Transform, we need only implement this circuit in the reverse order. This will be useful in the next section.

2.1.14 Quantum Phase Estimation

Quantum Phase Estimation (QPE) is one of the most useful actions we can perform with quantum computers, and as such it serves a key role in many quantum algorithms. The purpose of the algorithm, is:

Given a unitary matrix U, QPE will estimate the phase ϕ of $U|\psi\rangle = e^{2\pi i\phi}|\psi\rangle$. Our state $|\psi\rangle$ in this case is an eigenvector of U, and $e^{2\pi i\phi}$ is the corresponding eigenvalue.

If it is unclear why this is a useful thing to be able to do, that's alright, here we just introduce the algorithm, and we will look at an application of QPE in section 2.1.15.

In order to explore this algorithm, we will first address the intuition behind it, then look at the circuit which implements it, and finally provide the mathematical foundation for the algorithm.

Quantum Phase Estimation works in three main steps. The first is to use phase kickback (section 2.1.10) to write the phase of our gate U, in the Fourier basis, to t counting qubits. We then use the inverse Quantum Fourier Transform to transform this number from the Fourier basis, to the computational basis. Finally, we measure the t counting qubits, and calculate ϕ.

In order to understand how the phase is written to the counting qubits, we combine two previously discussed concepts, phase kickback, and counting in the Fourier basis. When counting in the Fourier basis using t qubits, we can represent any number between 0 and $2^t - 1$ as rotations around the z-axis of the Bloch Sphere. The most significant qubit (q_0) makes one full rotation about the Bloch Sphere when we count from 0 to $2^t - 1$, i.e. it rotates by $\frac{1}{2^t} \times 2\pi$ radians each time. The next qubit (q_1) will make *two* full rotations about the Bloch sphere. The one after that will make *four* full rotations, and after that *eight* full rotations etc. The least significant qubit will alternate between states $|+\rangle$ and $|-\rangle$ as we count from 0 to $2^t - 1$. In order to represent a number x between 0 and $2^t - 1$, the most significant qubit will rotate by $\frac{x}{2^n} \times 2\pi$ radians around the z-axis. The qubit after that will rotate by $\frac{2x}{2^n} \times 2\pi$ radians, the next by $\frac{4x}{2^n} \times 2\pi$ radians, the next by $\frac{8x}{2^n} \times 2\pi$ radians, and so on. A representation of the number 3 on 3 qubits is shown in figure 2.29.

Keeping this in mind, we address how phase kickback works into the algorithm. Recall that phase kickback means that the *control* qubit obtains a phase shift, or

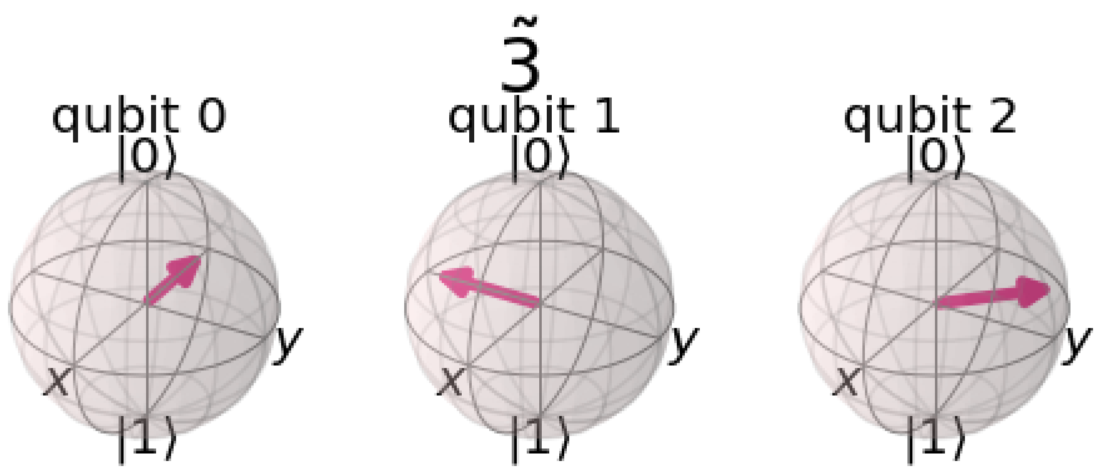


Figure 2.29: Representing the number 3 with three qubits in the Fourier basis on Bloch Spheres

rotation, while the target qubit does not. We start by taking our matrix in question U, and making a control gate out of it. When applied, the control qubit is now rotated proportionally to $e^{2\pi i\phi}$, the eigenvalue of U. This rotation is analogous to the $\frac{x}{2^n} \times 2\pi$ rotation applied in the counting example. Following the same pattern, each subsequent qubit controls the U gate twice as many times as the previous (i.e. 1 application, 2 applications, 4, 8, 16 etc..). In this way we encode our phase $e^{2\pi i\phi}$ as a number between 0 and $2^t - 1$ in the Fourier Basis.

After this encoding, we perform the inverse QFT to obtain that number in the computational basis. Finally, we can measure this state, and calculate the angle ϕ.

A general circuit that implements the Quantum Phase Estimation algorithm is shown in figure 2.30. Below we examine the mathematics behind the circuit.

At the start of the circuit, our state is $|\psi_0\rangle = |0\rangle^{\otimes t} |\psi\rangle$. Here $|0\rangle^{\otimes t}$ represents t qubits in the $|0\rangle$ state, and $|\psi\rangle$ is an eigenvector of U. We then perform the following steps:

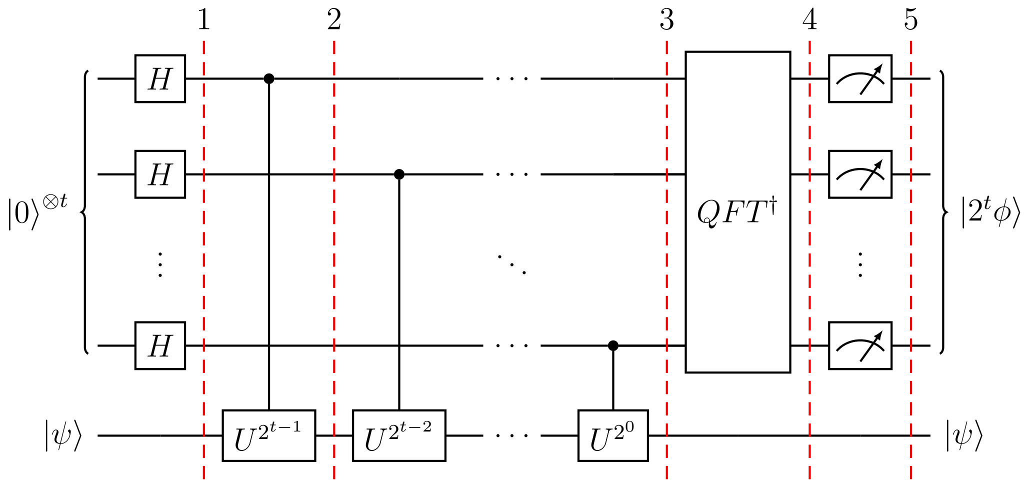


Figure 2.30: General circuit for implementing Quantum Phase Estimation using t counting qubits

1. Apply the H-gate to the t counting qubits. $H^{\otimes t}$ denotes the t qubit H-gate:

$$|\psi_1\rangle = H^{\otimes t}|\psi_0\rangle = \frac{1}{2^{\frac{t}{2}}}[(|0\rangle + |1\rangle)_0 \otimes (|0\rangle + |1\rangle)_1 \otimes \cdots \otimes (|0\rangle + |1\rangle)_{t-1}] \otimes |\psi\rangle \tag{2.148}$$

Here the subscript on each parenthesis denotes the specific counting qubit. We factored out the normalization factors $\frac{1}{\sqrt{2}}$ for each t qubit as $\frac{1}{2^{t/2}}$.

2. Apply the controlled U gate 2^{t-1} times to $|\psi\rangle$ with qubit 0 as the control:

$$|\psi_2\rangle = \frac{1}{2^{\frac{t}{2}}}\left[(|0\rangle + e^{2\pi i 2^{t-1}\phi}|1\rangle)_0 \otimes (|0\rangle + |1\rangle)_1 \otimes \cdots \otimes (|0\rangle + |1\rangle)_{t-1}\right] \otimes |\psi\rangle \tag{2.149}$$

In order to see what happened to qubit 0, we first recall that the gate U has eigenvector $|\psi\rangle$ and corresponding eigenvalue $e^{2\pi i\phi}$, so $U|\psi\rangle = e^{2\pi i\phi}|\psi\rangle$. We then observe that:

$$U^{2^{t-1}}|\psi\rangle = U^{2^{t-2}}U|\psi\rangle = U^{2^{t-2}}e^{2\pi i\phi}|\psi\rangle = \cdots = e^{2\pi i 2^{t-1}\phi}|\psi\rangle \tag{2.150}$$

It is due to phase kickback that the phase $e^{2\pi i 2^{t-1}\phi}$ is applied to qubit 0.

3. We then apply controlled U gates half as many times, with each subsequent qubit controlling it. Meaning; qubit 1 controls the application of 2^{t-2} U gates, the next qubit controls 2^{t-3} U gates, all the way down to qubit $t-1$, which only controls a single U gate:

$$|\psi_3\rangle = \frac{1}{2^{\frac{t}{2}}}\Big(|0\rangle + e^{2\pi i 2^{t-1}\phi}|1\rangle\Big)\otimes\cdots\otimes\Big(|0\rangle + e^{2\pi i 2^{1}\phi}|1\rangle\Big)\otimes\Big(|0\rangle + e^{2\pi i 2^{0}\phi}|1\rangle\Big)\otimes|\psi\rangle \tag{2.151}$$

In order to simplify $|\psi_3\rangle$, we first expand the tensor products of the t counting qubits. In doing so, we note that we will have states ranging from $|0\dots0\rangle$ to $|1\dots1\rangle$, representing the numbers 0 to 2^t-1 in binary. As for the coefficients of these states, we note that all the $|0\rangle$ states bring with them a coefficient of 1. Since all the coefficients are multiplied together when expanding the tensor product, this is a very convenient coefficient to have. We also note that each $|1\rangle$ state will bring with it its own $e^{2\pi i x\phi}$ coefficient, which will be multiplied with any other $e^{2\pi i x\phi}$ present. Here the value of x in the exponent depends on the qubit. For the most significant qubit, $x = 2^{t-1}$, and for the least significant qubit, $x = 2^0 = 1$. For example, if we look at the state $|0\rangle$, which in our case is represented as $|0\dots0\rangle$, we see that its coefficient is 1 since all the $|0\rangle$ states have coefficient 1. Looking at the state $|1\rangle$, represented as $|0\dots01\rangle$, we see that only the least significant qubit is a $|1\rangle$. From equation 2.151 we can see that the $|0\dots01\rangle$ state has coefficient $e^{2\pi i 2^0\phi}$. For the state $|2\rangle$, represented as $|0\dots010\rangle$, we see the only coefficient that is not 1 comes from the second to last qubit. From equation 2.151 we can see that the $|0\dots010\rangle$ state has coefficient $e^{2\pi i 2^1\phi}$. For the state $|3\rangle$, represented as $|0\dots011\rangle$, we see that our coefficient will be $e^{2\pi i 2^1\phi}\times e^{2\pi i 2^0\phi} = e^{2\pi i(2^0+2^1)\phi} = e^{2\pi i 3\phi}$. Looking at the final state $|2^t-1\rangle$, which is represented as $|1\dots 1\rangle$, we see that the coefficient will be

$$\prod_{k=0}^{t-1} e^{2\pi i 2^k \phi} = e^{2\pi i \left(\sum_{k=0}^{t-1} 2^k\right)\phi} \tag{2.152}$$

In order to simplify this, we can take advantage of the following rule [18]:

$$\sum_{k=0}^{t-1} 2^k = \sum_{k=1}^{t} 2^{k-1} = 2^t - 1 \tag{2.153}$$

Combining 2.152 and 2.153, we see that the coefficient of $|1\dots 1\rangle$ is $e^{2\pi i(2^t-1)\phi}$. Finally, if we observe the coefficient of state $|1\dots 10\rangle$, we see it will be the same as for $|1\dots 1\rangle$, except we don't multiply in the least significant qubits' coefficient, $e^{2\pi i 2^0 \phi}$. Thus the coefficient for $|1\dots 10\rangle$ will be $e^{2\pi i(2^t-2)\phi}$. Writing this all out, we see that:

$$\begin{aligned}|\psi_3\rangle = \frac{1}{2^{\frac{t}{2}}} \Big(& e^{2\pi i(2^t-1)\phi}|1\dots 1\rangle + e^{2\pi i(2^t-2)\phi}|1\dots 10\rangle + \dots + e^{2\pi i 3\phi}|0\dots 011\rangle \\ & + e^{2\pi i 2\phi}|0\dots 010\rangle + e^{2\pi i 1\phi}|0\dots 01\rangle + e^{0}|0\dots 0\rangle \Big) \otimes |\psi\rangle\end{aligned} \tag{2.154}$$

Replacing the values in the $|\ \rangle$'s with the decimal representation (instead of binary representation), we get:

$$\begin{aligned}|\psi_3\rangle = \frac{1}{2^{\frac{t}{2}}} \Big(& e^{2\pi i(2^t-1)\phi}\left|2^t-1\right\rangle + e^{2\pi i(2^t-2)\phi}\left|2^t-2\right\rangle + \dots + e^{2\pi i 3\phi}|3\rangle \\ & + e^{2\pi i 2\phi}|2\rangle + e^{2\pi i 1\phi}|1\rangle + e^{0}|0\rangle \Big) \otimes |\psi\rangle\end{aligned} \tag{2.155}$$

Note that the value in the exponent matches the value in the $|\ \rangle$. We see now that we can rewrite this as the sum:

$$|\psi_3\rangle = \frac{1}{2^{\frac{t}{2}}} \sum_{k=0}^{2^t-1} e^{2\pi i k\phi} |k\rangle \otimes |\psi\rangle \tag{2.156}$$

4. We note that the counting qubits in $|\psi_3\rangle$ are now in a state that looks very similar to the result of applying the Quantum Fourier Transform. We recall that the QFT acts on a state $|x\rangle$ as follows:

$$QFT\,|x\rangle = \frac{1}{2^{\frac{t}{2}}} \Big(|0\rangle + e^{\frac{2\pi i}{2}x}|1\rangle\Big) \otimes \Big(|0\rangle + e^{\frac{2\pi i}{2^2}x}|1\rangle\Big) \otimes \cdots \otimes \Big(|0\rangle + e^{\frac{2\pi i}{2^t}x}|1\rangle\Big) \tag{2.157}$$

If we replace x in equation 2.157 with $2^t\phi$, we exactly get equation 2.151. We can see that we were able to prepare our counting qubits in the state $|2^t\phi\rangle$ in the Fourier basis! In order to measure this state in the computational basis, we must first perform the inverse Quantum Fourier Transform. The inverse QFT on t qubits is given by:

$$QFT_t^\dagger\,|y\rangle = \frac{1}{2^{\frac{t}{2}}} \sum_{x=0}^{2^t-1} e^{-2\pi i\frac{xy}{2^t}} |x\rangle \tag{2.158}$$

The $QFT^\dagger$ is in this form because the QFT gate is unitary (as all gates must be 2.15), so in order to invert it, we take the complex transpose. We apply the inverse QFT to our state $|\psi_3\rangle$ as follows:

$$QFT_t^\dagger |\psi_3\rangle = \frac{1}{2^{\frac{t}{2}}} \sum_{k=0}^{2^t-1} e^{2\pi ik\phi} \left(\frac{1}{2^{\frac{t}{2}}} \sum_{x=0}^{2^t-1} e^{-2\pi i\frac{xk}{2^t}} |x\rangle \right) \otimes |\psi\rangle \tag{2.159}$$

$$= \frac{1}{2^t} \sum_{k=0}^{2^t-1} \sum_{x=0}^{2^t-1} e^{2\pi ik\phi} e^{-2\pi i\frac{xk}{2^t}} |x\rangle \otimes |\psi\rangle \tag{2.160}$$

$$= \frac{1}{2^t} \sum_{k=0}^{2^t-1} \sum_{x=0}^{2^t-1} e^{-2\pi ik\phi\left(-\frac{2^t}{2^t}\right)} e^{-2\pi i\frac{xk}{2^t}} |x\rangle \otimes |\psi\rangle \tag{2.161}$$

$$= \frac{1}{2^t} \sum_{k=0}^{2^t-1} \sum_{x=0}^{2^t-1} e^{-\frac{2\pi ik}{2^t}(-2^t\phi) - \frac{2\pi ik}{2^t}x} |x\rangle \otimes |\psi\rangle \tag{2.162}$$

$$|\psi_4\rangle = \frac{1}{2^t} \sum_{k=0}^{2^t-1} \sum_{x=0}^{2^t-1} e^{-\frac{2\pi ik}{2^t}(x-2^t\phi)} |x\rangle \otimes |\psi\rangle \tag{2.163}$$

In 2.159 we apply the inverse QFT to our counting qubit state $|k\rangle$. On line 2.160 we factor out the $\frac{1}{2^{t/2}}$ that came with the inverse QFT and combine it with the same factor from $|\psi_3\rangle$ to get $\frac{1}{2^t}$. We also move both sums together and multiply the contents together. On line 2.161 we take the value in the first exponent, and multiply it by -1 and $(-2^t/2^t)$, essentially multiplying it by 1 and making no change. We do this to help us reduce the equation. On line 2.162 we do some more re-arranging of the value in the first exponent, bringing the $1/2^t$ under the $2\pi ik$, and moving the ϕ into the parenthesis. We also combine the two exponentials by subtracting the value in the seconds exponent ($e^a e^{-b} = e^{a-b}$). Finally, on line 2.163, we factor out the $-\frac{2\pi ik}{2^t}$.

5. Here we measure the t counting qubits. We can see from equation 2.163 that if $x = 2^t\phi$ we will have the largest amplitude. This is because when $x = 2^t\phi$, the exponent in the sum is e^0, regardless of what the value of k is. Thus each of those amplitudes will directly add to each-other giving a total amplitude of $\frac{k}{2^t}$. For all other values of x, the value in the exponent will not be zero, meaning each vector will have a different phase. When summed, the amplitude

of these vectors can only ever be as big as $\frac{k}{2^t}$, but will often be less or even zero. Therefore, when we measure the state it is highly probable that we will measure:

$$|\psi_5\rangle = \left|2^t\phi\right\rangle \otimes |\psi\rangle \tag{2.164}$$

If $2^t\phi$ is an integer then we will get the above state with very high probability. However if it is not an integer, it has been shown that we will measure this state with probability greater than $4/\pi^2$ [16]. This is because x is an integer, so can only get arbitrarily close to $2^t\phi$. Additionally, the more qubits we add, the closer we get to estimating ϕ.

Here we have shown how quantum phase estimation works, and presented a quantum circuit that implements it. In the following section we will look at an application of quantum phase estimation.

2.1.15 Quantum Counting

The quantum counting algorithm is used to calculate the number of solutions that an instance of a Grover Oracle has [19]. This is important, because we need to know the number of solutions an Oracle has, so that we can determine how many times to apply our Grover operation in our circuit.

In order to see how this works, we first return to our graphical representation of what the Grover operator G does. This representation is shown again in figure 2.31. Each successive application of G rotates the state $|\psi\rangle$ by another θ radians in the basis created by $|\beta\rangle$ and $|\alpha\rangle$. We recall also that $|\alpha\rangle$ represents all the non-solutions to our problem, and $|\beta\rangle$ represents all the solutions. Therefore we can represent the Grover

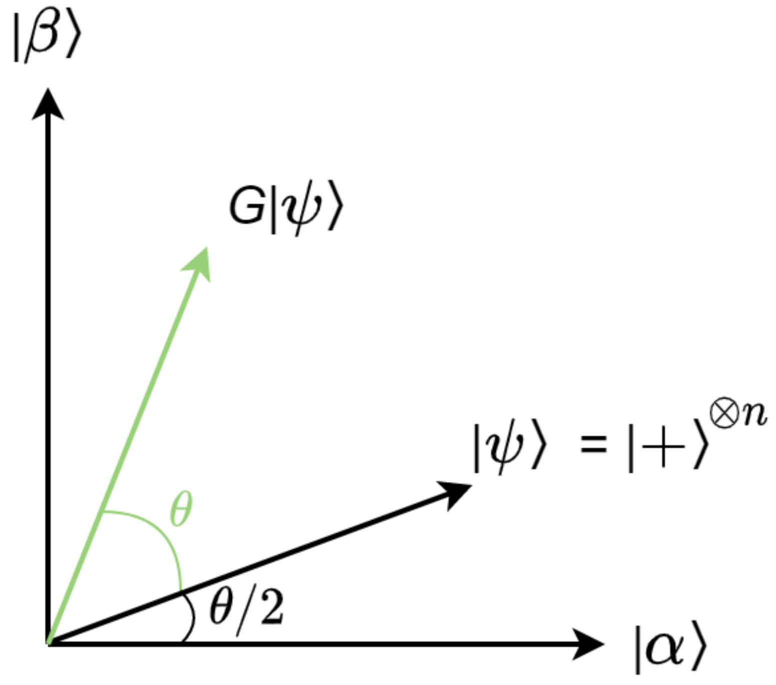


Figure 2.31: Visualization of Grover Operator applied to state $|\psi\rangle$

iteration G as follows:

$$G^k |\psi\rangle = \cos\left(\frac{2k+1}{2}\theta\right)|\alpha\rangle + \sin\left(\frac{2k+1}{2}\theta\right)|\beta\rangle \tag{2.165}$$

where k is the number of applications, and θ is as defined in figures 2.20 and 2.31. Because of this interpretation, we can now represent the Grover operator G as a *rotation* matrix:

$$G = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \tag{2.166}$$

We can verify that this is an accurate representation of G by applying it to the general state $|\psi\rangle = \cos\left(\frac{2k+1}{2}\theta\right)|\alpha\rangle + \sin\left(\frac{2k+1}{2}\theta\right)|\beta\rangle$:

$$G|\psi\rangle = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} \cos\left(\frac{2k+1}{2}\theta\right) \\ \sin\left(\frac{2k+1}{2}\theta\right) \end{bmatrix} = \begin{bmatrix} \cos\theta\cos\left(\frac{2k+1}{2}\theta\right) - \sin\theta\sin\left(\frac{2k+1}{2}\theta\right) \\ \sin\theta\cos\left(\frac{2k+1}{2}\theta\right) + \cos\theta\sin\left(\frac{2k+1}{2}\theta\right) \end{bmatrix} \tag{2.167}$$

We can then use the following identity,

$$\begin{aligned} \cos(a \pm b) &= \cos a \cos b \mp \sin a \sin b \\ \sin(a \pm b) &= \sin a \cos b \pm \sin b \cos a \end{aligned} \tag{2.168}$$

to arrive at the final vector:

$$G|\psi\rangle = \begin{bmatrix} \cos\left(\theta + \frac{2k+1}{2}\theta\right) \\ \sin\left(\theta + \frac{2k+1}{2}\theta\right) \end{bmatrix} \tag{2.169}$$

Now that we know we can represent G as a rotation matrix, we return to the discussion on Quantum Phase Estimation. Recall that the function of QPE was to find ϕ in $U|\psi\rangle = e^{2\pi i\phi}|\psi\rangle$, where $e^{2\pi i\phi}$ and $|\psi\rangle$ are eigenvalues and eigenvectors of U. We can observe that G has the eigenvectors:

$$\begin{bmatrix} -i \\ 1 \end{bmatrix}, \quad \begin{bmatrix} i \\ 1 \end{bmatrix} \tag{2.170}$$

with corresponding eigenvalues $e^{-i\theta}$ and $e^{i\theta}$, respectively. This can be verified by:

$$G\begin{bmatrix} i \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} i \\ 1 \end{bmatrix} = \begin{bmatrix} i\cos\theta - \sin\theta \\ i\sin\theta + \cos\theta \end{bmatrix} = \begin{bmatrix} (i)(\cos\theta + i\sin\theta) \\ \cos\theta + i\sin\theta \end{bmatrix} = e^{i\theta}\begin{bmatrix} i \\ 1 \end{bmatrix} \tag{2.171}$$

and

$$\begin{aligned} G\begin{bmatrix} -i \\ 1 \end{bmatrix} &= \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} -i \\ 1 \end{bmatrix} = \begin{bmatrix} -i\cos\theta - \sin\theta \\ -i\sin\theta + \cos\theta \end{bmatrix} = \begin{bmatrix} (-i)(\cos\theta - i\sin\theta) \\ \cos\theta - i\sin\theta \end{bmatrix} \\ &= e^{-i\theta}\begin{bmatrix} -i \\ 1 \end{bmatrix} \end{aligned} \tag{2.172}$$

Thankfully, we do not need to prepare our state $|\psi\rangle$ specially in either of these states. This is because when regarding our state $|\psi\rangle$ in the $|\alpha\rangle$ and $|\beta\rangle$ basis, it is actually in a superposition of the two eigenstates. To observe this fact, we do the following:

Let

$$|a\rangle \equiv \begin{bmatrix} i \\ 1 \end{bmatrix}, \quad |b\rangle \equiv \begin{bmatrix} -i \\ 1 \end{bmatrix} \tag{2.173}$$

In the basis formed by $|\alpha\rangle$ and $|\beta\rangle$, these states can be expressed as:

$$|a\rangle = i\,|\beta\rangle + |\alpha\rangle \tag{2.174}$$

$$|b\rangle = -i\,|\beta\rangle + |\alpha\rangle \tag{2.175}$$

Combining the two equations, we can solve for $|\alpha\rangle$ and $|\beta\rangle$, which lets us express those states as the eigenstates of their basis.

$$\begin{aligned} |a\rangle + |b\rangle &= (i\,|\beta\rangle + |\alpha\rangle) + (-i\,|\beta\rangle + |\alpha\rangle) = 2\,|\alpha\rangle \\ |\alpha\rangle &= \frac{1}{2}(|a\rangle + |b\rangle) \end{aligned} \tag{2.176}$$

$$\begin{aligned} |a\rangle - |b\rangle &= (i\,|\beta\rangle + |\alpha\rangle) - (-i\,|\beta\rangle + |\alpha\rangle) = 2i\,|\beta\rangle \\ |\beta\rangle &= \frac{1}{2i}(|a\rangle - |b\rangle) \end{aligned} \tag{2.177}$$

This lets us re-write our original state $|\psi\rangle = x\,|\alpha\rangle + y\,|\beta\rangle$ in terms of these two eigenstates:

$$|\psi\rangle = x\frac{1}{2}(|a\rangle + |b\rangle) + y\frac{1}{2i}(|a\rangle - |b\rangle) \tag{2.178}$$

$$= \frac{1}{2}(x - iy)\,|a\rangle + \frac{1}{2}(x + iy)\,|b\rangle \tag{2.179}$$

In order to make things less clunky, we define the following:

$$|\tilde{a}\rangle \equiv \frac{1}{2}(x - iy)\,|a\rangle\,, \quad \left|\tilde{b}\right\rangle \equiv \frac{1}{2}(x + iy)\,|b\rangle \tag{2.180}$$

Thus:

$$|\psi\rangle = |\tilde{a}\rangle + \left|\tilde{b}\right\rangle \tag{2.181}$$

If we apply the Grover operator G to this state, we will get:

$$G\,|\psi\rangle = G\,|\tilde{a}\rangle + G\left|\tilde{b}\right\rangle = e^{i\theta}\,|\tilde{a}\rangle + e^{-i\theta}\left|\tilde{b}\right\rangle \tag{2.182}$$

Now we can revisit the discussion on Quantum Phase Estimation to see how this change propagates through the equations. What we will first examine, is how the phase kickback will work with our controlled Grover operation. In order to see this, we will apply the controlled Grover operation to the state $|+\rangle\,|\psi\rangle$. I will denote the controlled Grover operation as G_c:

$$\begin{aligned} G_c\,|+\rangle\,|\psi\rangle &= G_c\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes (|\tilde{a}\rangle + \left|\tilde{b}\right\rangle) \\ &= G_c\frac{1}{\sqrt{2}}\left(|0\tilde{a}\rangle + \left|0\tilde{b}\right\rangle + |1\tilde{a}\rangle + \left|1\tilde{b}\right\rangle\right) \\ &= \frac{1}{\sqrt{2}}\left(|0\tilde{a}\rangle + \left|0\tilde{b}\right\rangle + G\,|1\tilde{a}\rangle + G\left|1\tilde{b}\right\rangle\right) \\ &= \frac{1}{\sqrt{2}}\left(|0\tilde{a}\rangle + \left|0\tilde{b}\right\rangle + e^{i\theta}\,|1\tilde{a}\rangle + e^{-i\theta}\left|1\tilde{b}\right\rangle\right) \\ &= \frac{1}{\sqrt{2}}\left((|0\rangle + e^{i\theta}\,|1\rangle)\,|\tilde{a}\rangle + (|0\rangle + e^{-i\theta}\,|1\rangle)\left|\tilde{b}\right\rangle\right) \end{aligned} \tag{2.183}$$

We notice that we end up with one term per eigenstate/eigenvalue pair. If we apply G_c to this state again, we get:

$$\begin{aligned}
&G_c \frac{1}{\sqrt{2}} \left(|0\tilde{a}\rangle + \left|0\tilde{b}\right\rangle + e^{i\theta} |1\tilde{a}\rangle + e^{-i\theta} \left|1\tilde{b}\right\rangle \right) \\
&= \frac{1}{\sqrt{2}} \left(|0\tilde{a}\rangle + \left|0\tilde{b}\right\rangle + Ge^{i\theta} |1\tilde{a}\rangle + Ge^{-i\theta} \left|1\tilde{b}\right\rangle \right) \\
&= \frac{1}{\sqrt{2}} \left(|0\tilde{a}\rangle + \left|0\tilde{b}\right\rangle + e^{i2\theta} |1\tilde{a}\rangle + e^{-i2\theta} \left|1\tilde{b}\right\rangle \right) \\
&= \frac{1}{\sqrt{2}} \left((|0\rangle + e^{i2\theta} |1\rangle) |\tilde{a}\rangle + (|0\rangle + e^{-i2\theta} |1\rangle) \left|\tilde{b}\right\rangle \right)
\end{aligned} \tag{2.184}$$

Using this result, we can explore what happens when we replace the general gate U with the Grover operator G in Quantum Phase Estimation. After the application of every controlled Grover operator, we will have a state very similar to that in equation 2.151:

$$\begin{aligned}
|\psi\rangle = \frac{1}{2^{\frac{t}{2}}} \Big[&\left(|0\rangle + e^{i2^{t-1}\theta} |1\rangle \right) \otimes \cdots \otimes \left(|0\rangle + e^{i2^{1}\theta} |1\rangle \right) \otimes \left(|0\rangle + e^{i2^{0}\theta} |1\rangle \right) \otimes |\tilde{a}\rangle \\
&+ \left(|0\rangle + e^{-i2^{t-1}\theta} |1\rangle \right) \otimes \cdots \otimes \left(|0\rangle + e^{-i2^{1}\theta} |1\rangle \right) \otimes \left(|0\rangle + e^{-i2^{0}\theta} |1\rangle \right) \otimes \left|\tilde{b}\right\rangle \Big]
\end{aligned} \tag{2.185}$$

We can use the same simplification process that we used to get equation 2.156 to express our state as:

$$\frac{1}{2^{\frac{n}{2}}} \sum_{k=0}^{2^n-1} e^{ik\theta} |k\rangle \otimes |\tilde{a}\rangle + e^{ik(2\pi-\theta)} |k\rangle \otimes \left|\tilde{b}\right\rangle \tag{2.186}$$

Note that we replaced $-\theta$ with $2\pi - \theta$. When we apply the inverse QFT to this state, we follow the same simplification as in equations 2.159-2.163:

$$QFT^\dagger \frac{1}{2^{\frac{n}{2}}} \sum_{k=0}^{2^n-1} e^{ik\theta} |k\rangle \otimes |\tilde{a}\rangle + e^{ik(2\pi-\theta)} |k\rangle \otimes \left|\tilde{b}\right\rangle \tag{2.187}$$

$$= \frac{1}{2^{\frac{t}{2}}} \sum_{k=0}^{2^t-1} e^{ik\theta} \left(\frac{1}{2^{\frac{t}{2}}} \sum_{x=0}^{2^t-1} e^{-2\pi i \frac{xk}{2^t}} |x\rangle \right) \otimes |\tilde{a}\rangle + e^{ik(2\pi-\theta)} \left(\frac{1}{2^{\frac{t}{2}}} \sum_{x=0}^{2^t-1} e^{-2\pi i \frac{xk}{2^t}} |x\rangle \right) \otimes \left|\tilde{b}\right\rangle \tag{2.188}$$

$$= \frac{1}{2^t} \sum_{k=0}^{2^t-1} \sum_{x=0}^{2^t-1} e^{ik\theta} e^{-2\pi i \frac{xk}{2^t}} |x\rangle \otimes |\tilde{a}\rangle + e^{ik(2\pi-\theta)} e^{-2\pi i \frac{xk}{2^t}} |x\rangle \otimes \left|\tilde{b}\right\rangle \tag{2.189}$$

$$= \frac{1}{2^t} \sum_{k=0}^{2^t-1} \sum_{x=0}^{2^t-1} e^{-ik\theta\left(-\frac{2^t 2\pi}{2^t 2\pi}\right)} e^{-2\pi i \frac{xk}{2^t}} |x\rangle \otimes |\tilde{a}\rangle + e^{-ik(2\pi-\theta)\left(-\frac{2^t 2\pi}{2^t 2\pi}\right)} e^{-2\pi i \frac{xk}{2^t}} |x\rangle \otimes \left|\tilde{b}\right\rangle \tag{2.190}$$

$$= \frac{1}{2^t} \sum_{k=0}^{2^t-1} \sum_{x=0}^{2^t-1} e^{-\frac{2\pi ik}{2^t}(-\frac{2^t}{2\pi}\theta) - \frac{2\pi ik}{2^t}x} |x\rangle \otimes |\tilde{a}\rangle + e^{-\frac{2\pi ik}{2^t}(-\frac{2^t}{2\pi}(2\pi-\theta)) - \frac{2\pi ik}{2^t}x} |x\rangle \otimes \left|\tilde{b}\right\rangle \tag{2.191}$$

$$= \frac{1}{2^t} \sum_{k=0}^{2^t-1} \sum_{x=0}^{2^t-1} e^{-\frac{2\pi ik}{2^t}(x - \frac{2^t}{2\pi}\theta)} |x\rangle \otimes |\tilde{a}\rangle + e^{-\frac{2\pi ik}{2^t}(x - \frac{2^t}{2\pi}(2\pi-\theta))} |x\rangle \otimes \left|\tilde{b}\right\rangle \tag{2.192}$$

Which can also be expressed as a superposition of likely states, where $x = \frac{2^t}{2\pi}\theta$ or $x = \frac{2^t}{2\pi}(2\pi - \theta)$:

$$\frac{1}{\sqrt{2}} \left(\left| \frac{2^n}{2\pi}\theta \right\rangle + \left| \frac{2^n}{2\pi}(2\pi - \theta) \right\rangle \right) \tag{2.193}$$

Notice that we can now apply QPE in order to obtain a value relative to θ! In order to calculate the θ that we are after, we examine the value measured from QPE, and what it means. We recall that the result of QPE will be the number $2^t\phi$, where ϕ is as in $U|\psi\rangle = e^{2\pi i\phi}|\psi\rangle$. The relationship between ϕ from QPE, and the θ desired from G is:

$$\theta = 2\pi\phi, \quad \phi = \frac{\theta}{2\pi} \tag{2.194}$$

The value x that we measure is:

$$x = 2^t \phi \tag{2.195}$$

Combining these two equations and solving for θ, we get:

$$x = 2^t \phi \tag{2.196}$$
$$x = 2^t \frac{\theta}{2\pi} \tag{2.197}$$
$$\theta = \frac{x}{2^t} 2\pi \tag{2.198}$$

With θ, we can now calculate the number of times we need to apply G in order to obtain our solution!

The above discussion presents the theory of the quantum counting algorithm. In practice, however, the solution calculation is slightly different. This is because of the way that the Diffuser is implemented for the Quantum Search algorithm. Recall from section 2.1.12 that our the Diffuser we implement has a global phase of (-1) that we were able to ignore for the Quantum Search. However, in the context of the Quantum Counting algorithm, we can no longer ignore this phase, as it is relative, not global. What this means is that when performing the Quantum Phase Estimation with the Grover operator, each application will come with an additional phase of $-1 = e^{i\pi}$. Looking back at the general form of QPE, we saw that we estimated ϕ in $U|\psi\rangle = e^{2\pi i\phi}|\psi\rangle$. In order to account for the relative phase shift added by our Diffuser, we assume that the general gate U also applies that phase shift. This gives us

$$U|\psi\rangle = e^{2\pi i\phi} e^{i\pi}|\psi\rangle = e^{2\pi i(\phi+1/2)}|\psi\rangle \tag{2.199}$$

This shift will propogate through all the equations, resulting in the final state that we measure being $|2^t(\phi + 1/2)\rangle$. So in order to calculate the angle θ we want to know for Grover's algorithm, $G|\psi\rangle = e^{\pm i\theta}|\psi\rangle$, we adjust equations 2.149-2.164 to take into

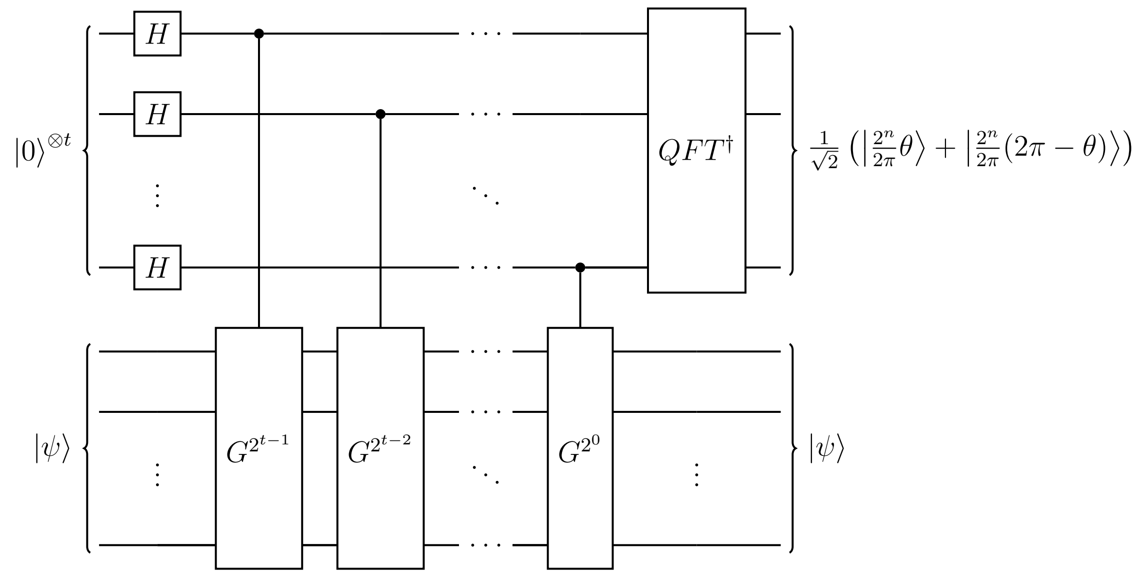

$|0\rangle^{\otimes t}$
H
H
H
$QFT^\dagger$
$\frac{1}{\sqrt{2}}\left(\left|\frac{2^n}{2\pi}\theta\right\rangle + \left|\frac{2^n}{2\pi}(2\pi-\theta)\right\rangle\right)$
$|\psi\rangle$
$G^{2^{t-1}}$
$G^{2^{t-2}}$
G^{2^0}
$|\psi\rangle$

Figure 2.32: General circuit that implements Quantum Counting algorithm for t counting qubits and n work qubits

account the phase shift. Assuming x is the value we measure from the Quantum Counting algorithm we have:

$$x = 2^t(\phi + 1/2) \tag{2.200}$$

$$x = 2^t\left(\frac{\theta}{2\pi} + 1/2\right) \tag{2.201}$$

$$\frac{x}{2^t} = \frac{\theta}{2\pi} + 1/2 \tag{2.202}$$

$$\theta = 2\pi\left(\frac{x}{2^t} - 1/2\right) \tag{2.203}$$

In order to create the circuit that implements this algorithm, we make G a controlled operation, and control it with the counting qubits used in QPE. The qubits that G acts on are prepared in the same way as for running Grover's algorithm. A circuit diagram showing this implementation is shown in figure 2.32.

2.2 Complexity Theory

In this section, I give a brief outline of complexity theory as it relates to classical computations. The majority of this section is presenting the methods with which we can analyze the complexity of a quantum algorithm.

In broad strokes, the idea of complexity theory is to determine the amount of resources required to solve a given problem. These resources are divided into physical resources (i.e. physical space required) and time resources (how long it takes to solve). In this paper I will be focusing on the time complexity of problems.

2.2.1 Big Oh Notation

The core idea to analyzing the time complexity of a given problem or algorithm, is determining how the number of operations required will scale with the size of the input. To help with this, we use what is called "Big-Oh" notation, which looks like $O(\cdot)$. The value in the parenthesis is what we are primarily concerned about. The smaller this value, the better, because this means it takes less time to solve that specific problem. For example, if we were trying to determine if a number n is even or odd, we can simply check if it is evenly divisible by 2. It doesn't matter how big or small the number n is, it will always take the same amount of time to determine if it is even or odd. This is an example of a function that as a time complexity of $O(1)$. Now let's imagine that we are tasked with adding up all the values in a list of length n. Here we can see that the time it takes to compute the sum depends on the length of the list. For every additional value in the list, we have to perform another addition operation. Since the number of operations required scales linearly with the size of our input (n), we say that this function has a time complexity of $O(n)$. Figure

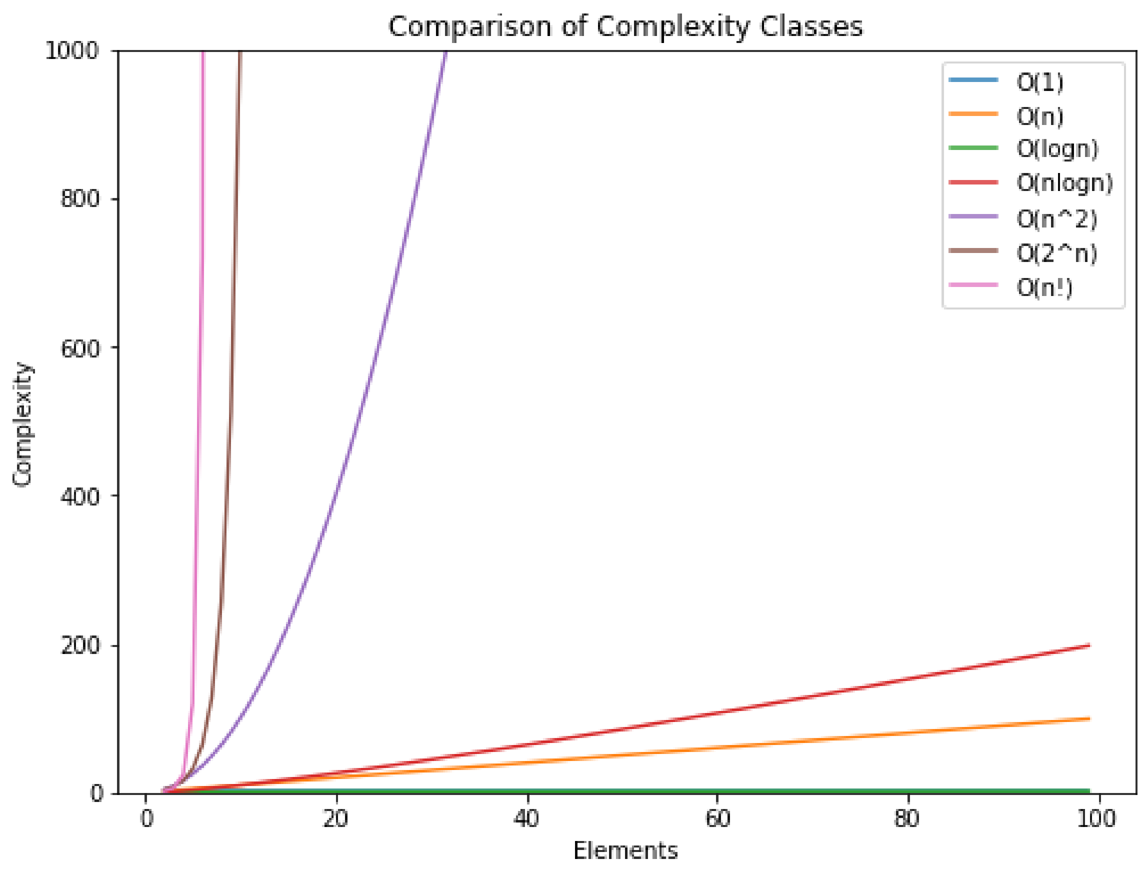


Figure 2.33: Complexities of different $O(\cdot)$ functions. Note that $O(logn)$ is covering $O(1)$.

2.33 shows how the time it takes to solve a problem scales with the input size, for different time complexities.

2.2.2 P Problems

There exists a group of problems in complexity theory that are in the complexity class **P**, where **P** stands for Polynomial time. In order for a problem to be in class **P**, it need only be deterministically solvable in polynomial time. This means that a computer can find a solution to the given problem in time $O(n^k)$, where k is any real number. These problems are generally considered to be "quickly" solvable, and as such are considered a boon among computer scientists.

2.2.3 NP Problems

Another complexity class is that of **NP**, or Non-Polynomial time. Problems in **NP** are called decision problems. Decision problems are problems that have a yes or no answer. A problem is in **NP** if there exists a polynomial time algorithm for checking a solution to that problem. For example, imagine we are trying to split a list of numbers into two lists, such that the sums of those lists are equal. This problem may not be easy to solve, but it is easy (and fast) to check a solution to the problem; simply compute the sum of both lists and compare.

2.2.4 NP-Complete Problems

NP-Complete problems are a subset of **NP** problems for which there are no known polynomial-time solutions. In order for a problem to be in the class **NP**-Complete, it must satisfy the following two constraints:

1. Be in the class **NP**
2. All other problems in **NP**-Complete are reducible to it in polynomial time.

We already know how to check the first constraint, so I will briefly discuss the second constraint. To be "reducible" basically means we can transform the given problem (problem A) to another problem (problem B), solve problem B, then transform that solution back into a solution for problem A. Figure 2.34 shows a diagram of this. In order to be reducible in polynomial time, the functions $f(\cdot)$ and $h(\cdot)$ need to run in polynomial time. Additionally, in figure 2.34, problem A is the problem we are trying to show is in **NP**-Complete.

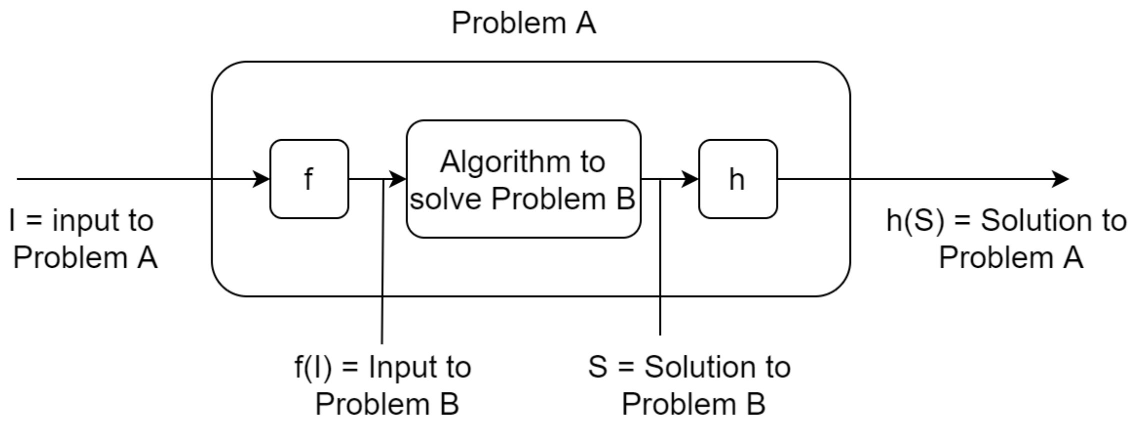

Problem A
f
Algorithm to solve Problem B
h
I = input to Problem A
h(S) = Solution to Problem A
f(I) = Input to Problem B
S = Solution to Problem B

Figure 2.34: Problem reduction. Here Problem A is reduced to Problem B

2.2.5 NP-Hard Problems

NP-Hard problems are informally, at least as hard as the hardest problems in **NP**. In general, these take the form of optimization problems. For example the chromatic number problem is **NP**-Hard. While it is a hotly contested question of if **P** = **NP**, a generally accepted picture of how these complexity classes interact is shown in 1.1.

2.2.6 BPP Problems

Bounded-error probabilistic polynomial time, **BPP** problems differ from the previously discussed classes in that there is randomness involved. **BPP** problems are decision problems that are solvable on a probabilistic computer in polynomial time with error probability less that 1/3. That is, if the solution to a given decision problem is "yes", then a **BPP** algorithm will output "yes" at least 2/3rds of the time. The three constraints that must be true for a problem to be in **BPP** are:

1. It must run in polynomial time
2. It can make random decisions.

3. It must give the correct answer to a given problem with probability at least 2/3, regardless of whether the answer is "yes" or "no".

2.2.7 BQP Problems

The **BQP** is different than the previous complexity classes in that it strictly deals with problems run on a *quantum* computer, as opposed to a classical computer. Aside from this one difference, **BQP** is defined the same as **BPP**. An image presenting how these complexity classes are related is shown in figure 2.35. For an in depth survey of quantum complexity, see [20].

2.2.8 Complexity Analysis of Quantum Algorithms

When analyzing quantum algorithms, we primarily look at the quantum circuit itself, and perform the analysis at the gate level. Before diving further into the complexity analysis, we need to introduce the concept of circuit depth. From [21], "The depth of a circuit is the number of distinct timesteps at which gates are applied." Figure 2.36 shows three different circuits that will help illustrate the concept of circuit depth. The circuit in figure 2.36a has a depth of 1, since all 3 X-gates can be executed at the same time. The circuit in figure 2.36b has a depth of 2 since the H-gate must be applied *after* the X-gate on qubit 0. Finally, the circuit in figure 2.36c also has a depth of 2 for the same reason as 2.36b. When performing complexity analysis, we focus on how the depth of a circuit grows with respect to the input size. With this in mind, we can use the familiar Big-oh notation when talking about quantum algorithms.

Now that we have a concept of circuit depth, there is one more idea we need to introduce before calculating the time complexity of a quantum algorithm. We need

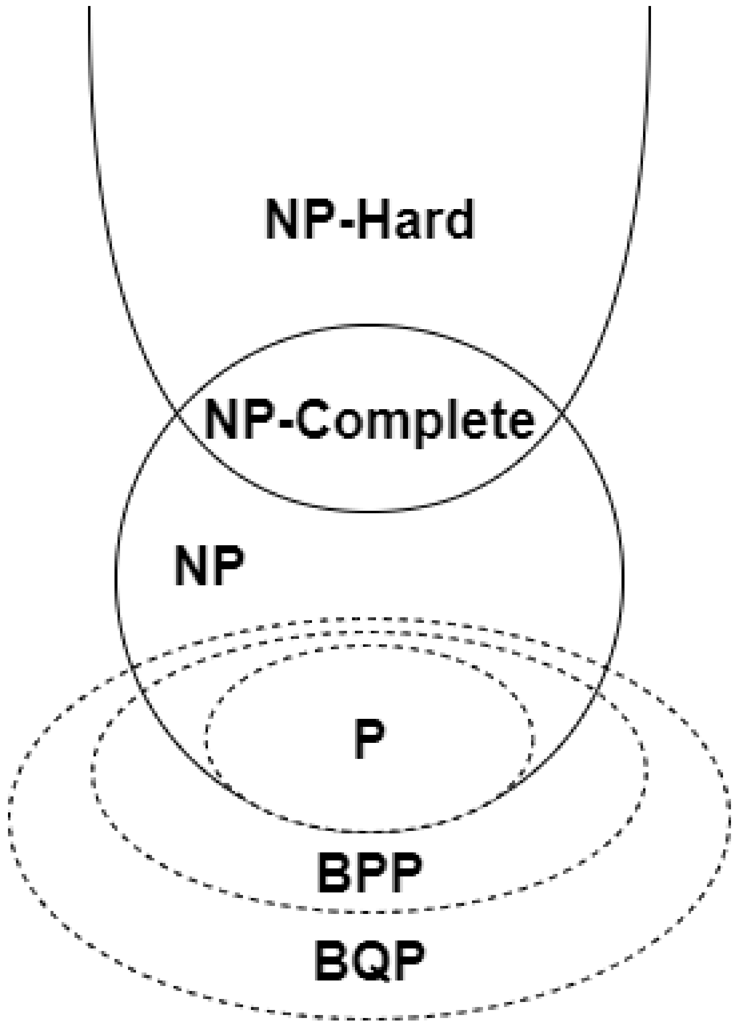
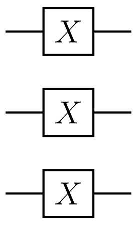
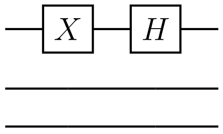


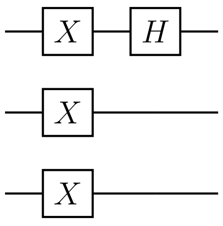
Figure 2.35: Potential interaction between Complexity Classes



(a) Example circuit with depth 1



(b) Example circuit with depth 2



(c) Example circuit with depth 2

Figure 2.36: Example circuits demonstrating what circuit depth represents

to create a universal set of gates, each of which we define to run in constant time. Recall that a universal set of gates means we can create any gate we want from the original set. The universal quantum gate set that I use for complexity analysis in this paper is called the Clifford+Z_N group, originally proposed by [22]. The gates in the Clifford group are; cNOT, X, Y, Z, H, and S. We have already seen all of these except for the S-gate, which is defined below:

$$S = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix} \tag{2.204}$$

The Clifford+Z_N group contains the addition of the Z_N gate, which is essentially an N^{th} root version of the Z-gate. The Z_N gate is defined below:

$$Z_N = \begin{bmatrix} 1 & 0 \\ 0 & e^{\frac{i\pi}{N}} \end{bmatrix} \tag{2.205}$$

where N is a parameter that we can pick.

Now that we have picked a universal gate group, we can perform complexity analysis of a quantum algorithm. As we move through the circuit, any time we see any gate from the Clifford+Z_N group, we assign that a computation time of $O(1)$. Any time we see any gate that is *not* in the Clifford+Z_N group, we must first decompose it into gates from the Clifford+Z_N group, then the running time of the original gate is the depth of the decomposed circuit. This is more clearly seen when we look at a multi-controlled Toffoli gate (MCT). The authors of [22] showed that it is possible to decompose an MCT gate with n controls into a circuit of depth n. Thus we take the runtime of an n-MCT gate to be $O(n)$. With this decomposition, we can perform complexity analysis of the algorithm presented in this paper.

Chapter 3

RELATED WORK

Over the past few years, there has been much work on applying Grover's algorithm to a wide range of NP-Complete problems. The most popular problem of study is the Satisfiability problem [23, 24, 11, 25, 12, 26]. There are two main reasons for this. The first is that the Satisfiability problem was the first computer problem that was provably difficult to solve, and laid the foundation for Karp's seminal paper introducing the original 21 NP-Complete problems [10]. The other reason is that the Satisfiability has a fairly direct mapping to an implementation of Grover's algorithm. The key step in applying Grover's algorithm to a specific problem is the creation of the Oracle. The Oracle's job is to determine what is a solution, and what isn't a solution. In creating an Oracle for Satisfiability, we can pretty much directly map the individual clauses to a circuit element, and string these elements together. Because of these reasons, the application of Grover's algorithm to Satisfiability appears in major quantum computing textbooks [11, 12].

There have also been applications of Grover's algorithm to a variety of other NP-Complete problems. These include, Vertex Cover [27], Edge Cover [24], Max Clique [28, 29], K-Coloring [30, 31], and Subset Sum [32] to name a few. All these approaches essentially boil down to creating a problem-specific Oracle, as the rest of Grover's algorithm is general.

Another area of interest has been in creating a generalized NP-Complete problem solver that takes advantage of the speed up offered by Grover's algorithm. In [33], the authors propose just such a framework. They first present an Oracle for the

Satisfiability problem, and use Grover's algorithm with this Oracle to solve an instance of the Satisfiability problem. Then they go on to build a framework that will reduce any instance of an NP-Complete problem into an instance of Satisfiability, then apply their quantum algorithm to obtain the solution. This is a beneficial approach because it does not require the user have any previous knowledge of quantum computing and quantum algorithms. However it does have some drawbacks in that the quantum circuits created for problems other than Satisfiability are often inefficient in that they use an excess of qubits and/or quantum gates. This poses a problem for near term applications because modern day quantum computers have less than 100 qubits and thus will be unable to run complex circuits. Additionally, the deeper the circuit (i.e. the more gates there are), the more prone the system is to error. Because of this it is useful to analyze each NP-Complete problem separately, in order to design efficient Oracles for each.

This is what the authors of [30] have done. They designed an efficient Oracle to solve the K-Coloring problem. Specifically, they compared their solution to the given Oracle in [33] for a 3-Coloring problem. In [33] the solution required $n * k$ data qubits, and $O((n * k)^2)$ ancilla qubits, where n is the number of nodes in the graph, and k is the number of colors. In contrast, the solution presented in [30] requires only $n * \lceil log_2 k \rceil$ data qubits, and $O(n)$ ancilla qubits. This is clearly a large improvement over that of [33], and allows the K-Coloring to be solved on near term quantum devices.

There is another algorithm that is being explored for applications to computationally difficult problems, specifically optimization problems. That is the Quantum Approximate Optimization Algorithm, or QAOA [34]. As its name suggests, it provides an approximate solution to a given optimization problem. This is in stark contrast to the previous works discussed, because they were solving non-optimization problems. Another difference between this algorithm and previous works, is that it provides an approximate solution, so there is no guarantee that the solution provided is the optimal solution.

Chapter 4

IMPLEMENTATION

In this section, I outline the details of my implementation, and provide the algorithms for generating the required quantum circuits to solve the K-Coloring problem, and the Chromatic Number problem.

It is worth noting that these algorithms were all implemented on IBM's IBMQ platform. This platform always initializes qubits in state $|0\rangle$, so this is the initial state for all qubits in this section.

4.1 Problem Definitions

Here I define the two problems that are solved in this paper. The first is the K-Coloring problem, and the second is the Chromatic Number problem. The Chromatic Number problem was selected for study specifically because there are currently no applications of quantum algorithms to the Chromatic Number problem. As mentioned in section 3, the Quantum Search algorithm has been applied to a number of NP-Complete problems, including the K-Coloring problem [30]. However, these applications have not been extended to the optimization versions of these problems. That is what this paper provides through the use of the Quantum Counting algorithm.

Another reason that these problems were chosen, is because of the tangible real-world applications that these problems have. One of the most common applications of the K-Coloring and Chromatic Number problems is in scheduling. For this application, vertices can be modeled as events to be scheduled, and potential conflicts can be modeled as edges between vertices. Thus a minimum coloring will schedule all the events with no conflicts and optimal overlap of events. Another application is in mobile radio frequency assignment, i.e. how to assign frequencies to neighboring towers such that they don't overlap with each other.

The K-Coloring problem is defined as follows:

K-Coloring

Input: A graph $G = (V, E)$ and integer k. V is the set of all vertices, and E is the set of all edges in graph G.

Output: The function $\phi()$, where ϕ is defined as: $\phi : V \to \mathbb{Z}_k$ such that if u and v are adjacent then $\phi(u) \neq \phi(v)$

The Chromatic Number problem is defined as follows:

Chromatic Number

Input: A graph $G = (V, E)$

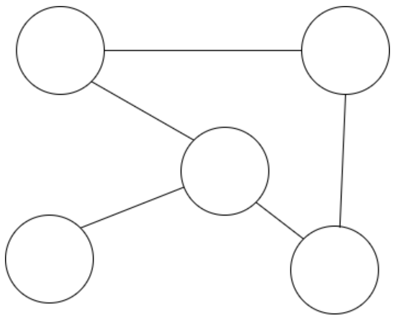
Output: The chromatic number k of graph G. The chromatic number of a graph is defined as the smallest number of colors needed to color the vertices of graph G such that no adjacent vertices are the same color [35].

An example of the K-Coloring problem is shown in figure 4.1.

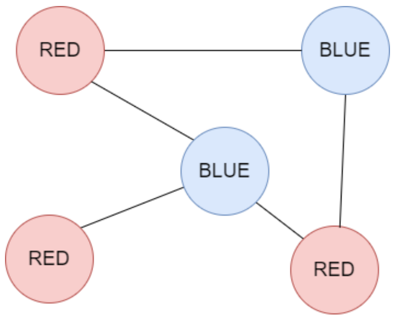
With these problem definitions in mind, we can go about solving them.

4.2 General Approach

The main purpose of this paper is to present a quantum algorithm that solves the Chromatic Number problem faster than the best classical algorithms. In order to solve



(a) Original Graph



(b) One possible solution to K-Coloring problem

Figure 4.1: An example graph and coloring for 2 colors. Note that the chromatic number of this graph is 2

the Chromatic Number problem, the Quantum Counting algorithm is used. Recall from section 2.1.15 that the purpose of the Quantum Counting algorithm is to count the number of solutions that a specific Oracle has for a given problem. The solution to the Chromatic Number problem is the smallest number of colors with which we can color the vertices of graph G such that no adjacent vertices are the same color. Put another way, the solution of Chromatic Number is the smallest k for which the K-Coloring problem has a solution.

The approach taken to solve Chromatic Number is to generate K-Coloring Oracles for different values of k, and find the smallest value of k for which the corresponding Oracle has at least one solution. So, an Oracle is built for the input graph G, and a specific value k. Then the Quantum Counting circuit is built using this Oracle, and the number of solutions is calculated after running the corresponding circuit. A searching technique called binary search is used to find the smallest value of k that produces an Oracle with solutions to the K-Coloring problem for graph G. Binary search finds a target element in a sorted list by first checking the middle value of the list. If the value at the middle is larger than the target value, we limit our search to the larger half of the list, else we limit our search to the lower half of the list. This process repeats until we find the target value. When applying this to Chromatic

Number, we are looking for the smallest k that produces solutions, i.e. a value of $k-1$ will have 0 solutions to the K-Coloring problem.

With this process in mind, we first discuss how to create an Oracle that solves the K-Coloring problem.

4.3 Quantum Oracle for K-Coloring

The Oracle itself, which is what solves the K-Coloring problem, is taken from [30]. Using the quantum counting algorithm, we can determine if an Oracle for a given number k has any solutions. With this approach, we can systematically find the smallest k that has solutions to the given problem, and that will be the Chromatic Number of our input graph G.

In this section, I describe how the Oracle from [30] works, and present an algorithm that will generate this circuit for any given instance of the K-Coloring problem. The generation algorithm is taken from [30], with some slight modifications.

In order to construct an Oracle for a specific instance of the K-Coloring problem, the first step is to determine the number of qubits required. We can do so from the number of vertices in the graph $G = (V, E)$, and the number of colors k. We first let $|V| = n$. We then calculate the number of qubits required to represent k distinct colors, which is $\lceil log_2 k \rceil$. We use $\lceil log_2 k \rceil$ qubits because n qubits can represent 2^n distinct values, thus to represent k distinct values we need to find a value x such that $2^x \geq k$, which we do by taking the log base 2 of k. Therefore, in order to fully represent a solution to this instance of K-Coloring, we need $n \times \lceil log_2 k \rceil$ data qubits. In the following discussions, we will let I_i represent the color of a single vertex, so I_i is represented by $\lceil log_2 k \rceil$ qubits. By placing $m = n \times \lceil log_2 k \rceil$ qubits in a superposition, we can represent all possible solutions at the same time. Initially, these data qubits are in the state, $|\xi\rangle = |0\rangle^{\otimes m}$. Next, in order to carry out the calculations required, we need n ancilla qubits, $|\theta\rangle = |0\rangle^{\otimes n}$. The purpose of these ancilla qubits will be described in the following sections. We also potentially need one additional ancilla qubit if we need to perform the Invalid Color Detector (described later), which we prepare as $|\zeta\rangle = |0\rangle$. Finally, we have a single output qubit, $|\phi\rangle = |0\rangle$. We can describe the entire initial state of the Oracle as:

$$|\psi_0\rangle = |\xi\rangle \otimes |\theta\rangle \otimes |\zeta\rangle \otimes |\phi\rangle = |0\rangle^{\otimes m} \otimes |0\rangle^{\otimes n} \otimes |0\rangle \otimes |0\rangle \tag{4.1}$$

After the declaration of these qubits, we initialize them in the desired states for the rest of the algorithm. Since we want to process all possible solutions at once, we will initialize the data qubits in the equal superposition state $|\xi\rangle = |+\rangle^{\otimes m}$. This can be done by applying the H-gate to each qubit. We also want to initialize the ancilla qubits $|\theta\rangle$ in the $|1\rangle$ state, which can be accomplished by applying the X-gate to each ancilla qubit. Finally, we want our output qubit $|\phi\rangle$ to be in the state $|-\rangle$, which can be accomplished by applying the X-gate, then the H-gate. This initialization step brings us to the state:

$$|\psi_1\rangle = |+\rangle^{\otimes m} \otimes |1\rangle^{\otimes n} \otimes |0\rangle \otimes |-\rangle \tag{4.2}$$

With this state, we can move onto the two main logical components of the K-Coloring Oracle; Invalid Color Detector (ICD), and comparators.

4.3.1 Invalid Color Detector

The purpose of the Invalid Color Detector is restrict our search space to answers that use at most k colors. The reason this is used is because we represent each unique

”color” as a binary number, encoded into a fixed number of qubits. It is very likely that we will be able to represent more colors than we need. For example if we have 5 colors, then we need at least 3 qubits to represent them. But with 3 qubits, we can represent 8 unique values. If we don’t do anything to restrict this, the Oracle will produce solutions that use up to 8 colors, which in our case would be invalid solutions since $k = 5$. Thus we need to make sure that the Oracle can only use 5 of the possible 8 colors. We will let 0-4 (or in binary 000-100) represent our valid colors. Then the invalid colors are 101, 110, and 111. The Invalid Color Detector will make sure that none of these colors will appear in any of our answers. This is represented as the following equation:

$$ICD(I_0, I_1, \ldots, I_{n-1}, \zeta) = \begin{cases} \zeta = 0 & \text{if } I_0 or I_1 or \ldots or I_{n-1} \text{ is an invalid color} \\ \zeta = 1 & \text{if all colors are valid} \end{cases} \tag{4.3}$$

Here $I_0, I_1, \ldots, I_{n-1}$ represent the color of vertices $0, 1, \ldots, n-1$ where each I_i can be one or more qubits. In our example, each I_i is three qubits. The general circuit for an Invalid Color Detector is shown in figure 4.2. The implementation requires $n+1$ ancilla qubits, n for the qubits in $|\theta\rangle$ and 1 qubit in $|\zeta\rangle$, where n is the number of vertices. The ∂-ICD gate applied to each vertex I_i. ∂-ICD denotes a partial Invalid Color Detector, which detects all invalid colors for a single vertex. A concrete example of this is given below.

We implement the ∂-Invalid Color Detector with a multi-controlled Toffoli gate, and single qubit X-gates (if needed). The goal here is to activate the multi-controlled Toffoli gate if our data qubits are in any of the invalid color states. Continuing our example from before, in order to invalidate the color 111, we create a multi-controlled Toffoli gate with the three data qubits as the controls, and the corresponding ancilla qubit as the target. In other words, I_i is the control, while A_i is the target. For the

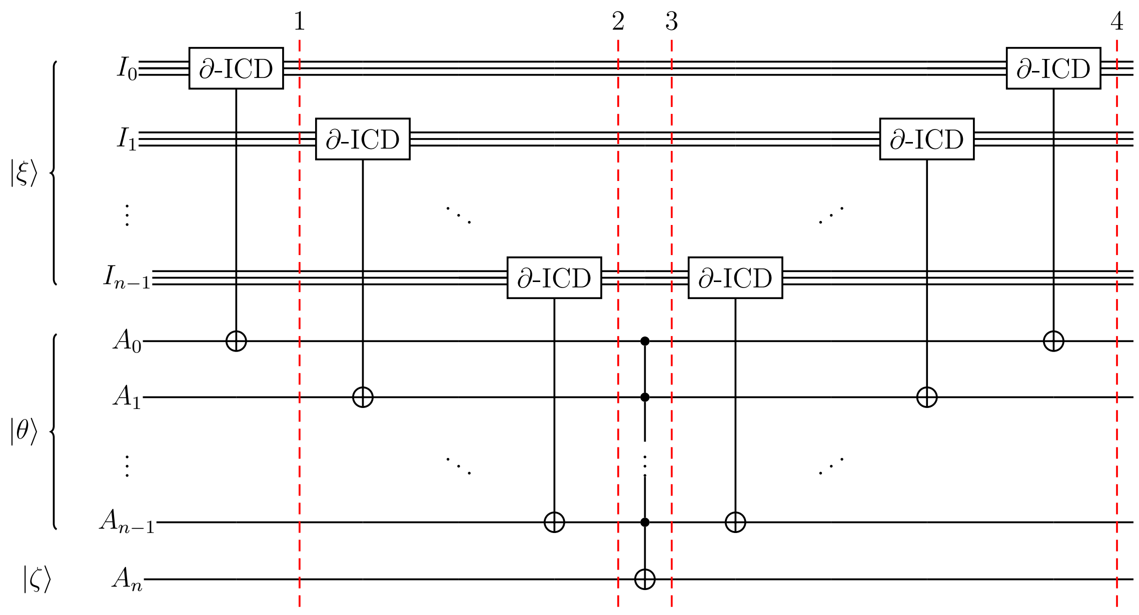


Figure 4.2: General circuit for Invalid Color Detector for n vertices. Note each I_i is the color of vertex i, and may be represented by 1 or more qubits. ∂-ICD denotes a partial Invalid Color Detector, which detects all invalid colors for a single vertex

110 color however, we need to first flip the last qubit from a 0 to a 1, *then* apply our multi-controlled Toffoli gate, then we flip the last qubit back to a 0. The same logic applies to color 101. The ∂-Invalid Color Detector in our example is shown in figure 4.3. We can describe this operation mathematically as

$$\partial\text{-ICD}(I_i, \theta_k) = \begin{cases} \theta_k = 0 & \text{if } I_i \text{ is an invalid color} \\ \theta_k = 1 & \text{if } I_i \text{ is a valid color} \end{cases} \tag{4.4}$$

We observe that the MCT will be activated and the ancilla qubit will be *flipped* in any case where I is an invalid color. By initializing the ancilla qubits $A_0, \dots A_{n-1}$ in the state $|1\rangle$, the circuit 4.3 implements equation 4.4.

We apply the same process to each vertex I_i and corresponding ancilla A_i in our encoding of the graph G. Once we have done this for all n vertices, we apply a

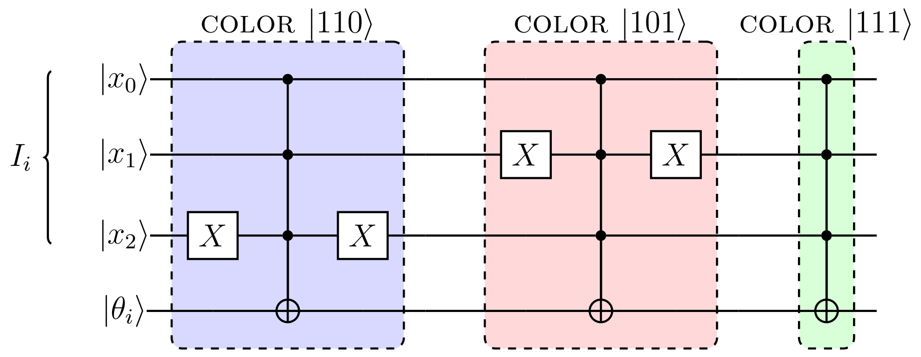


Figure 4.3: Example implementation of partial Invalid Color Detector for a single vertex I_i, with colors represented by qubits x_0, x_1, x_2. The state $|\theta_i\rangle$ is the output state, and is the ancillary qubit that corresponds to vertex i. Invalid colors are $|110\rangle, |101\rangle$, and $|111\rangle$

multi-controlled Toffoli gate controlled by ancilla qubits $A_0, \dots A_{n-1}$, and targeting A_n. The target qubit $|\zeta\rangle$ now holds the output of the full Invalid Color Detector. $|\zeta\rangle$ will be in state $|0\rangle$ if *any* of the vertices $I_0, \dots I_{n-1}$ are an invalid color, and $|1\rangle$ otherwise, satisfying equation 4.3. To see why this is true, we can step through the circuit in figure 4.2 for our three qubit example:

1. Apply the circuit in 4.3 to I_0 and A_0. Initially, I_0 and A_0 are described by

$$\begin{aligned} |I_0\rangle |A_0\rangle &= |+\rangle^{\otimes 3} |1\rangle \\ &= \frac{1}{\sqrt{8}} \Big(|000\rangle |1\rangle + |001\rangle |1\rangle + |010\rangle |1\rangle + |011\rangle |1\rangle + |100\rangle |1\rangle \\ &\qquad + |101\rangle |1\rangle + |110\rangle |1\rangle + |111\rangle |1\rangle \Big) \end{aligned} \tag{4.5}$$

Applying equation 4.4 to this state, recalling that the invalid colors are $|101\rangle$, $|110\rangle$ and $|111\rangle$, we get:

$$\begin{aligned} &\frac{1}{\sqrt{8}} \Big(|000\rangle |1\rangle + |001\rangle |1\rangle + |010\rangle |1\rangle + |011\rangle |1\rangle + |100\rangle |1\rangle + |101\rangle |0\rangle \\ &\qquad + |110\rangle |0\rangle + |111\rangle |0\rangle \Big) \\ &= \frac{1}{\sqrt{8}} \Big(|\text{valid colors}\rangle \otimes |1\rangle + |\text{invalid colors}\rangle \otimes |0\rangle \Big) \end{aligned} \tag{4.6}$$

2. Repeat the previous step for all other vertices. We observe that for any state containing an invalid color at least one ancilla qubit $A_0, \dots A_{n-1}$ will be $|0\rangle$

3. Apply the MCT with qubits $A_0, \dots A_{n-1}$ as the controls, and qubit A_n as the target. If any of the qubits $A_0, \dots A_{n-1}$ are $|0\rangle$, then the MCT will not activate, and qubit A_n will remain in state $|0\rangle$. If all the qubits $A_0, \dots A_{n-1}$ are $|1\rangle$ (i.e. all colors are valid), then qubit A_n gets flipped to state $|1\rangle$. Thus equation 4.3 is satisfied.

4. Un-compute the ancilla qubits $A_0, \dots A_{n-1}$, returning them all to state $|1\rangle$.

4.3.2 Comparators

Once the Invalid Color Detector has been applied, we apply a set of comparators. The comparator compares the colors of two vertices to check if they are equal. We can describe this operator mathematically as:

$$Comp(I_i, I_j, \theta_k) = \begin{cases} \theta_k = 0 & \text{if } I_i = I_j \\ \theta_k = 1 & \text{if } I_i \neq I_j \end{cases} \tag{4.7}$$

The general circuit implementing equation 4.7 with each I represented as c qubits is shown in figure 4.4. To see how circuit 4.4 implements equation 4.7, we will use two examples. For the first example, we let $I_i = |00\rangle$ and $I_j = |01\rangle$. For the second example, we let $I_i = |01\rangle$ and $I_j = |01\rangle$. In practice, the qubit f will be one of the ancilla qubits $A_0, \dots A_{n-1}$, which are initially in state $|1\rangle$. The circuit implementing the comparator for our examples is shown in figure 4.5. For our first example, we have

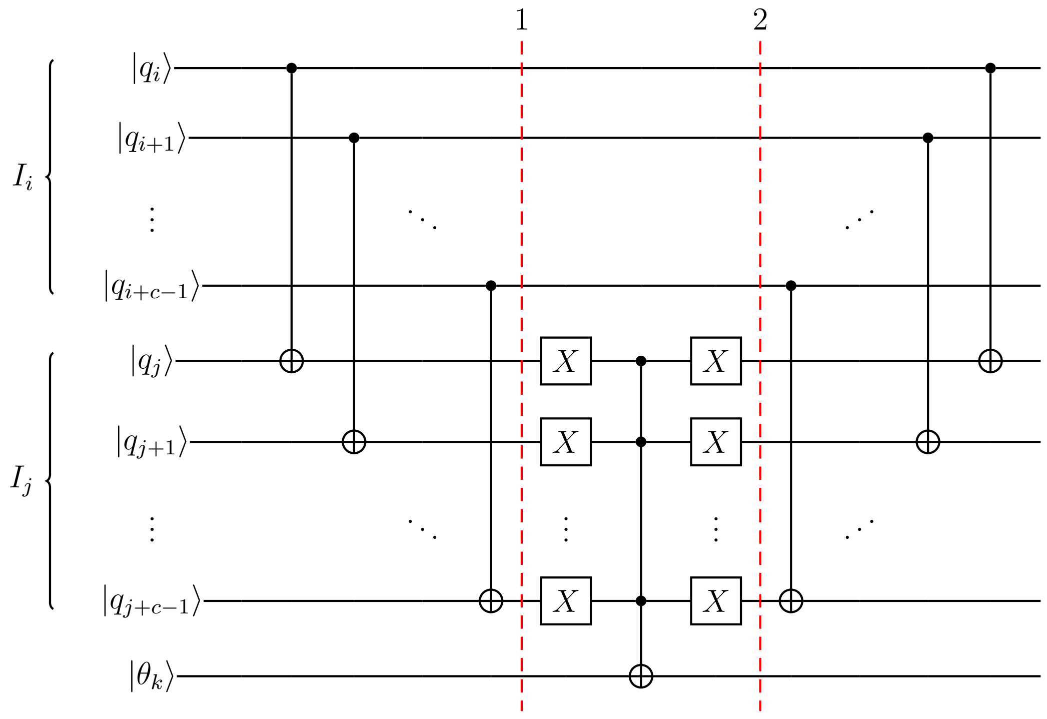


Figure 4.4: General Comparator circuit for vertices represented by variable c qubits.

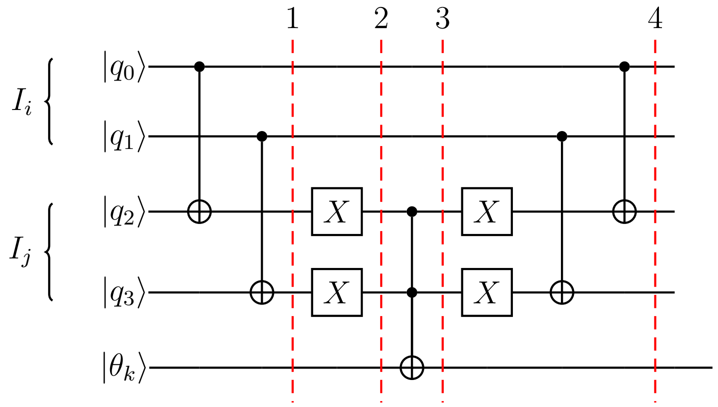


Figure 4.5: Comparator circuit for vertices I_i and I_j. Each vertex is represented by two qubits.

1. Apply the two cNOT gates to qubits q_2, q_3 controlled by qubits q_0, q_1 respectively. Our initial state is

$$|\psi_0\rangle = |I_i\rangle |I_j\rangle |\theta_k\rangle = |00\rangle |01\rangle |1\rangle \tag{4.8}$$

   After the cNOT gates, we have the state

$$|\psi_1\rangle = |00\rangle |01\rangle |1\rangle \tag{4.9}$$

   Since both controls are in state $|0\rangle$, neither cNOT was activated.

2. Apply X-gates to qubits q_2 and q_3.

$$|\psi_2\rangle = |00\rangle |10\rangle |1\rangle \tag{4.10}$$

3. Apply the MCT to qubit θ_k, controlled by qubits q_2 and q_3.

$$|\psi_3\rangle = |00\rangle |10\rangle |1\rangle \tag{4.11}$$

   Since $q_3 = |0\rangle$, qubit θ_k remains unchanged

4. Un-compute qubits q_2 and q_3:

$$|\psi_4\rangle = |00\rangle |01\rangle |1\rangle \tag{4.12}$$

Here we see that when $I_i \neq I_j$, our output qubit θ_k remains in state $|1\rangle$. In the next example we examine the case when $I_i = I_j$:

1. Apply the two cNOT gates to qubits q_2, q_3 controlled by qubits q_0, q_1 respectively. Our initial state is

$$|\psi_0\rangle = |I_i\rangle |I_j\rangle |\theta_k\rangle = |01\rangle |01\rangle |1\rangle \tag{4.13}$$

   After the cNOT gates, we have the state

$$|\psi_1\rangle = |01\rangle |00\rangle |1\rangle \tag{4.14}$$

2. Apply X-gates to qubits q_2 and q_3.

$$|\psi_2\rangle = |01\rangle |11\rangle |1\rangle \tag{4.15}$$

3. Apply the MCT to qubit θ_k, controlled by qubits q_2 and q_3.

$$|\psi_3\rangle = |01\rangle |11\rangle |0\rangle \tag{4.16}$$

   Since $q_2 = q_3 = |1\rangle$, qubit θ_k gets flipped to $|0\rangle$

4. Un-compute qubits q_2 and q_3:

$$|\psi_4\rangle = |01\rangle |01\rangle |0\rangle \tag{4.17}$$

Here we see that qubit θ_k is now a $|0\rangle$. This satisfies equation 4.7. When we apply the comparator block to our Oracle, we implement a comparator between every vertex in the graph G that share an edge. In order to simplify the number of gates needed to build the full Oracle, we define three different variants of the Comparator circuit. The first of these is the Full-Comparator, which is exactly as shown in figure 4.4. The second of which is the Open-Comparator, which is the Full-Comparator, but without

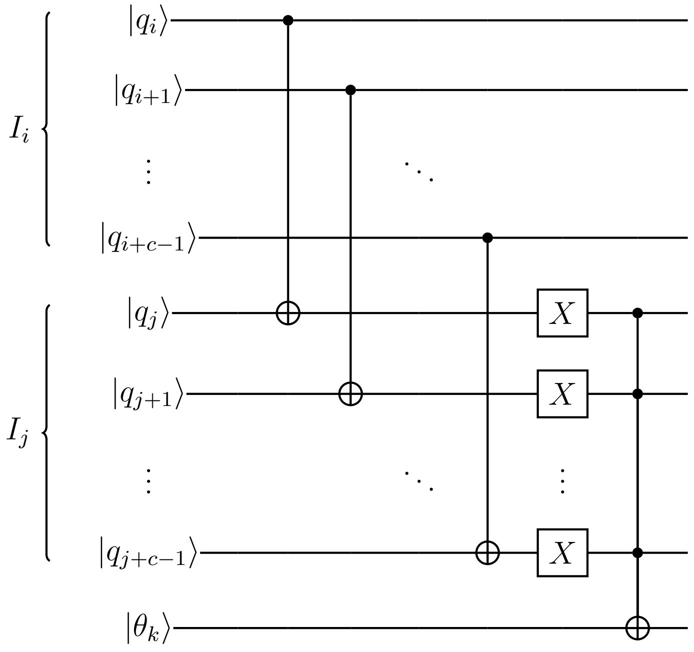


Figure 4.6: General Open-Comparator circuit for vertices represented by variable c qubits.

the uncomputation step. A general Open-Comparator is shown in figure 4.6. The other variation is the Close-Comparator, which is essentially the uncomputation part of the Full-Comparator. The general Close-Comparator circuit is shown in figure 4.7. The uses of these circuits will be discussed below.

Putting these pieces together, we create a general block diagram for the K-Coloring Oracle, shown in figure 4.8. In this circuit, we first initialize our qubits so they are in the proper states. Then we apply the Invalid Color Detector. After the Invalid Color Detector, the state of qubit $|\zeta\rangle$ will be $|1\rangle$ for states where none of $I_0, \dots I_{n-1}$ are invalid colors, and $|0\rangle$ otherwise. Next apply the comparator block. After the comparator block, the ancilla qubits $|\theta\rangle$ will be in the state $|\theta\rangle = |1\rangle^{\otimes n}$ iff all connected vertices in graph G are a different color. Finally, we apply an MCT to the output qubit $|\phi\rangle = |-\rangle$, controlled by qubits $|\theta\rangle$ and $|\zeta\rangle$. If all of the control qubits are in the $|1\rangle$ state, then $|\phi\rangle$ becomes $X|\phi\rangle = X|-\rangle = -|-\rangle$. This gives the entire state

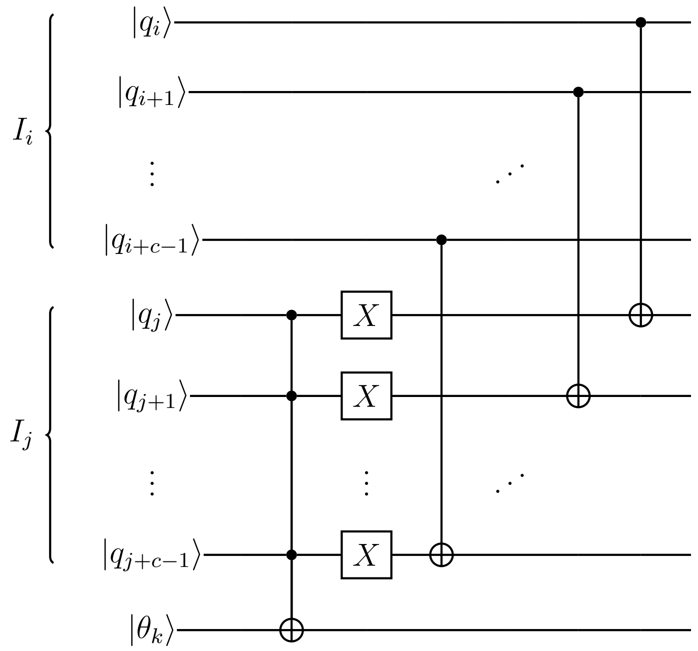


Figure 4.7: General Close-Comparator circuit for vertices represented by variable c qubits.

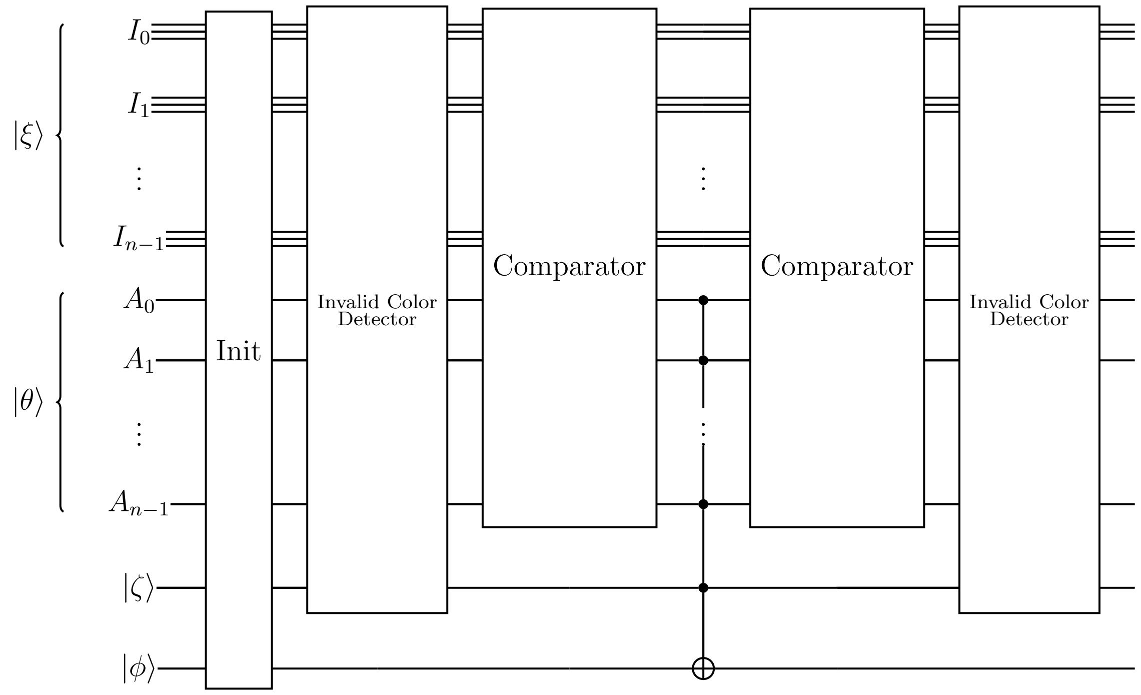


Figure 4.8: Block diagram for K-Coloring Oracle

$|\xi\rangle \otimes |\theta\rangle \otimes |\zeta\rangle \otimes |\phi\rangle$ a minus sign. We then apply the comparator block again, followed by the Invalid Color Detector to uncompute qubits $|\xi\rangle$, $|\theta\rangle$, and $|\zeta\rangle$.

4.4 Building The Quantum Oracle

Here I present a modified version of the algorithm from [30], which will generate the K-Coloring Oracle described in the previous section. The notable changes added in this paper are:

- Algorithm returns a circuit C instead of QASM code
- The use of Open, and Close-Comparators. This simplifies the circuit and reduces the overall number of gates required.
- The addition of an X-gate on line 23. This is a requirement, without the X-gate the algorithm does not work.
- The loop in 27 is reversed (from n to i instead of i to n) to mirror the loop on line 18.
- meaning of m variable is changed to simplify and clarify variables.

Algorithm 1 AutoGenOracleK-Coloring($G(V, E), k, init$)

```
Input: Graph $G(V,E)$, $|V| = n$, $|E| = e$, where $V$ is set of vertices and $E$ is set of
    edges. $k$ is an integer, representing the number of colors with which the graph
    should be colored. $init$ is a boolean, determining whether or not the qubits should
    be initialized.
Output: Quantum Circuit $C$ implementing Oracle to tag solutions to K-Coloring
    problem for $G(V,E)$, $k$.
 1: Let $I_r$ represent vertices where $0 \leq r \leq n-1$, $I_i$ corresponds to vertex $v_i$, and $I_i$
    is represented by $\lceil log_2 k \rceil$ qubits. $I_r$ requires $n * \lceil log_2 k \rceil$ qubits
 2: Let $A_r$ represent ancilla qubits where $0 \leq r \leq n-1$. $A_r$ requires $n$ qubits
 3: Let $A_{r+1}$ be a single qubit, required iff Invalid Color Detector is required
 4: Let $\phi$ be the output qubit
 5: Let $C$ be a circuit containing qubits $I_r, A_r, A_{r+1}, \phi$
 6: if $init$ is True then
 7:     Set $I_r$ data qubits as $|+\rangle$
 8:     Set $A_r$ qubits as $|1\rangle$
 9:     Set $A_{r+1}$ qubit as $|0\rangle$ iff Invalid Color Detector required
10:     Set $\phi$ qubit as $|-\rangle$
11: Add Invalid Color Detector (if required) for all possible invalid colors with $A_{r+1}$
    as target to $C$
12: $f = 0$, $l = n-1$
13: for $i = 0 \ldots n-1$ do
14:     $t = f$
15:     $type$ = "Full"
16:     if Vertex $i$ has degree $> 1$ then
17:         $type$ = "Open"
18:     for $j = i+1 \ldots n$ do
19:         if There is an edge($e$) connecting vertices $i$ and $j$ then
20:             Add a $type$-Comparator circuit with data qubits $I_i$ and $I_j$ as controls
                and $A_t$ as target to $C$
21:             $t = t+1$
22:     if $t > f+1$ then
23:         Add an X gate on qubit $A_l$ to $C$
24:         Add MCT gate with $A_f, \ldots A_t$ as controls and $A_l$ as target to $C$
25:         $l = l-1$
26:         $m = t-1$
27:         for $j = n-1 \ldots i$ do
28:             if There is an edge($e$) connecting vertices $i$ and $j$ then
29:                 Add a Close-Comparator circuit with data qubits $I_i$ and $I_j$ as con-
                    trols and $A_m$ as target to $C$
30:                 $m = m-1$
31:     else if $t = f+1$ then
32:         $f = f+1$
33: Add MCT gate with all ancillas as controls and $\phi$ as target to $C$
34: Add all gates applied during lines 12-32 to $C$ in reverse order
35: Repeat line 11
36: Return $C$
```

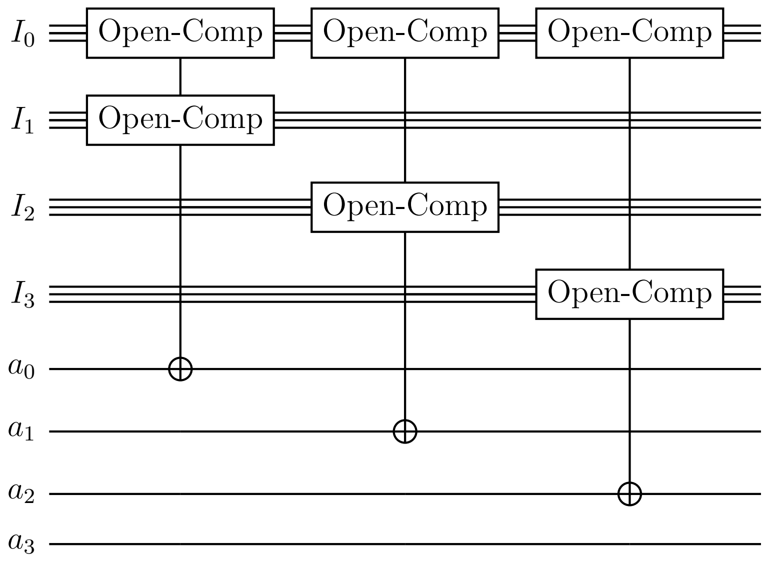


Figure 4.9: Circuit that compares (I_0, I_1), (I_0, I_2), and (I_0, I_3), writing results to a_0, a_1, and a_2 respectively

In this algorithm, lines 1-12 are setup, and application of the Invalid Color Detector. We then loop through each vertex in the graph, and lines 18-21 add a comparator between each vertex pair that has an edge between them. We notice that the type of this comparator changes based on how many edges vertex i has. If there is only one edge, then we add a Full-Comparator. However when vertex i has more than one edge, we only implement an Open-Comparator. This is because on lines 23-30, we will uncompute the affected qubits by adding Close-Comparators. We do this because it is more efficient than adding two Full-Comparators, which ends up wasting computations. The most unclear parts of this algorithm are the if statement on line 22 and the following code block, lines 23-30. The purpose of this code, is to utilize the concept of uncomputation in order to save on the number of ancilla qubits required to construct the Oracle. A careful observer may recognize that a graph $G = (V, E)$ can have up to $n(n-1)/2$ edges in it. Additionally, we see that lines 18-21 add a comparator for every edge in the graph, where each comparator (i.e. each edge) has a unique ancilla qubit that holds the answer. This implies that we need $n(n-1)/2$ ancilla qubits (one for each edge), instead of the n that we actually have. The code on lines 23-30 are what allow this Oracle to only need n ancilla qubits. Essentially what we are doing, is compressing the information about one vertex compared to all other vertices in the graph, into a single qubit. For example, lets imagine we have a fully connected graph with 4 vertices. On our first iteration of loop 13, we see that by the time we get to line 22 we have added a comparator between vertex 1, and vertices 2, 3, and 4. This is shown in figure 4.9. Note that in this circuit diagram, the "Comp" blocks are equivalent to the circuit in figure 4.4, and are condensed for readability. We notice that ancilla bits a_0, a_1, and a_2 hold all the comparison information for vertex I_0. Qubit a_0 holds $comp(I_0, I_1)$, qubit a_1 holds $comp(I_0, I_2)$, and qubit a_2 holds $comp(I_0, I_3)$. What the code on lines 23-30 does is encode all that information into ancilla qubit a_3, then un-compute ancillas a_0, a_1, and a_2 so that they no longer hold any important information, and can be safely re-used later on in the circuit. This is shown in figure 4.10. We note that an X-gate is placed on a_3 before the MCT. Ancillas a_0, a_1, and a_2 are described by:

$$|a_0\rangle = \begin{cases} 1 & \text{if } I_0 \neq I_1 \\ 0 & \text{else} \end{cases}, \quad |a_1\rangle = \begin{cases} 1 & \text{if } I_0 \neq I_2 \\ 0 & \text{else} \end{cases}, \quad |a_2\rangle = \begin{cases} 1 & \text{if } I_0 \neq I_3 \\ 0 & \text{else} \end{cases} \tag{4.18}$$

and qubit a_3 is described by

$$|a_3\rangle = |a_0 \cdot a_1 \cdot a_2\rangle \tag{4.19}$$

We can see from equations 4.18 and 4.19 that a_3 will be in state $|1\rangle$ only if all the vertices are different colors. We note that in figure 4.10 a_3 has an X-gate applied to it before the MCT gate. This is because the MCT gate will only activate if all the colors are not equal (which is what we want). However, since all the ancilla qubits start in state $|1\rangle$, activation of the MCT will actually flip a_3 from a $|1\rangle$ to a $|0\rangle$. Therefore we add the X-gate so that $|a_3\rangle$ is only in state $|1\rangle$ if the MCT gate is triggered.

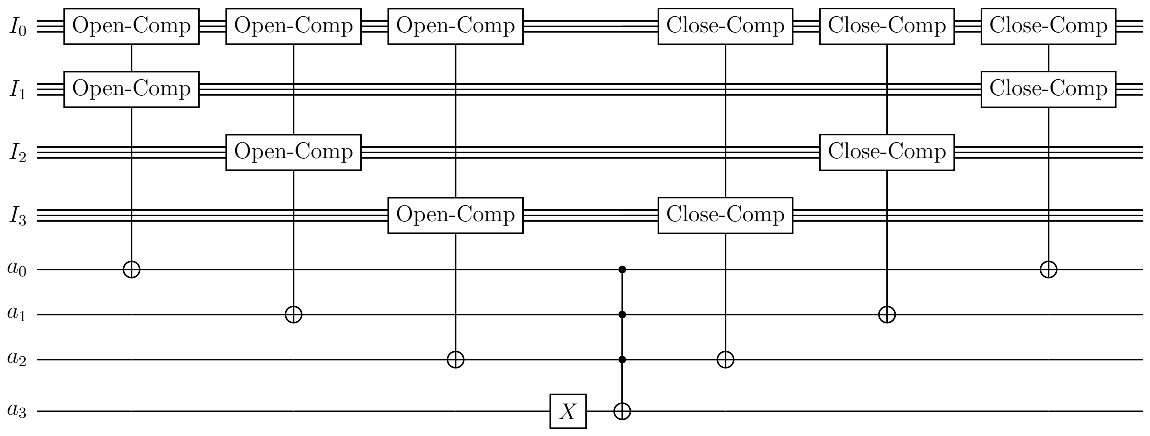


Figure 4.10: Full comparator for comparing colors of vertex 0 to all other vertices in a fully connected graph with 4 vertices

4.5 Solving Chromatic Number

In this section, I present two algorithms. The first is an implementation of the Quantum Counting algorithm specifically for a given K-Coloring Oracle. This algorithm will count the number of solutions an Oracle for a specific value of k has. The next algorithm puts all the pieces together to actually solve the Chromatic Number problem. Essentially what it does is a binary search through possible values of k for a given graph $G = (V, E)$. It is looking for the smallest value of k for which there are solutions to the K-Coloring problem. This value k is the Chromatic Number, and is returned by the algorithm.

Algorithm 2 COUNTK-COLORING($G(V, E), k, t$)

```
Input: Graph $G(V,E)$, $|V| = n$, $|E| = e$, where $V$ is set of vertices and $E$ is set of
    edges. $k$ is an integer, representing the number of colors with which the graph
    should be colored. $t$ is the number of counting qubits to be used for Quantum
    Counting.
Output: The number of solutions to K-Coloring problem for $G(V,E)$, $k$.
 1: $Oracle$ =AUTOGENORACLEK-COLORING($G(V,E), k, False$)
 2: Let $\omega$ be the number of qubits in $Oracle$
 3: Add Diffuser to qubits $0, \ldots n\lceil log_2 k\rceil - 1$ of $Oracle$
 4: Let $C$ be a circuit with $\omega + t$ qubits and $t$ classical bits
 5: Add H-gates for qubits $0, \ldots t - 1$ to $C$
 6: Add initialization sequence for AUTOGENORACLEK-COLORING on qubits
    $t, \ldots t + \omega - 1$ to $C$
 7: $iters = 1$
 8: for $q = 0 \ldots t - 1$ do
 9:     for $i = 0 \ldots iters$ do
10:         Add $Oracle$ applied to qubits $t, \ldots t + \omega - 1$, controlled by qubit $q$, to $C$
11:     $iters = iters \times 2$
12: Add QFT$^\dagger$ applied to qubits $0, \ldots t - 1$ to $C$
13: Add measurements for qubits $0, \ldots t - 1$ to classical bits $0, \ldots t - 1$ to $C$
14: Run $C$, and let $m$ be the value of classical bits $0, \ldots t - 1$
15: $\theta = 2\pi \left(\frac{m}{2^t} - \frac{1}{2}\right)$
16: $M = 2^n \sin^2 \theta/2$
17: Return $M$
```

Algorithm 3 CHROMATICNUMBER($G(V, E)$)

```
Input: Graph $G(V,E)$, $|V| = n$, $|E| = e$, where $V$ is set of vertices and $E$ is set of
    edges.
Output: The chromatic number of $G(V,E)$.
 1: if $e = 0$ then
 2:     Return 1
 3: $k_{max} = n, \quad k_{min} = 2$
 4: while $k_{min} \le k_{max}$ do
 5:     $k_{guess} = \lfloor \frac{k_{max}+k_{min}}{2} \rfloor$
 6:     $M$ = COUNTK-COLORING($G(V,E), k_{guess}, \lceil n/2 \rceil + 3$)
 7:     if $M = 0$ then
 8:         $k_{min} = k_{guess} + 1$
 9:     else
10:         $k_{max} = k_{guess} - 1$
11: if $M = 0$ then
12:     Return $k_{guess} + 1$
13: Return $k_{guess}$
```

The number of counting qubits t used in the ChromaticNumber algorithm is taken from [16]. This selection of $\lceil n/2 \rceil + 3$ counting qubits will give us a correct solution with probability of at least 5/6. That is to say, if $M = 0$, then the CountK-Coloring algorithm will return 0 with probability 5/6. If $M \neq 0$, then CountK-Coloring will return a non-zero answer with probability 5/6.

4.6 Solving K-Coloring

Solving the K-Coloring problem requires direct application of Grover's algorithm. After building the specific Oracle for the given problem using AutoGenOracleK-Coloring, the next step is to add the Diffuser to the $n\lceil log_2 k\rceil$ qubits that represent the vertex colors. This creates a full Grover operator. Once the Grover operator is constructed, the next step is to calculate the number of times to apply it in order to achieve the maximum likelihood of measuring a solution state. This calculation is given in equation 2.72. The number of solutions and the angle θ can be obtained by using CountK-Coloring. After calculating the number of iterations required, a quantum circuit must be built which consists of the initialization, followed by the required number of Oracle and Diffuser applications, followed by a measurement of the data qubits. This process is shown in figure 4.11. The resulting measurement of the data qubits will be a solution to the given K-Coloring problem with probability at least 0.12 [16]. This probability can be improved to close to 1 by a few repetitions of the counting-search procedure. The solution will be in the following form. Let x_i denote vertex i from graph $G = (V, E)$, and $c(x_i)$ denote the color of vertex i. Note that $c(x_i)$ is represented by $\lceil log_2 k\rceil$ qubits, where k is the number of colors used. Then the solution will be in the form

$$|\psi\rangle = |c(x_0)c(x_1)\dots c(x_{n-1})\rangle \tag{4.20}$$

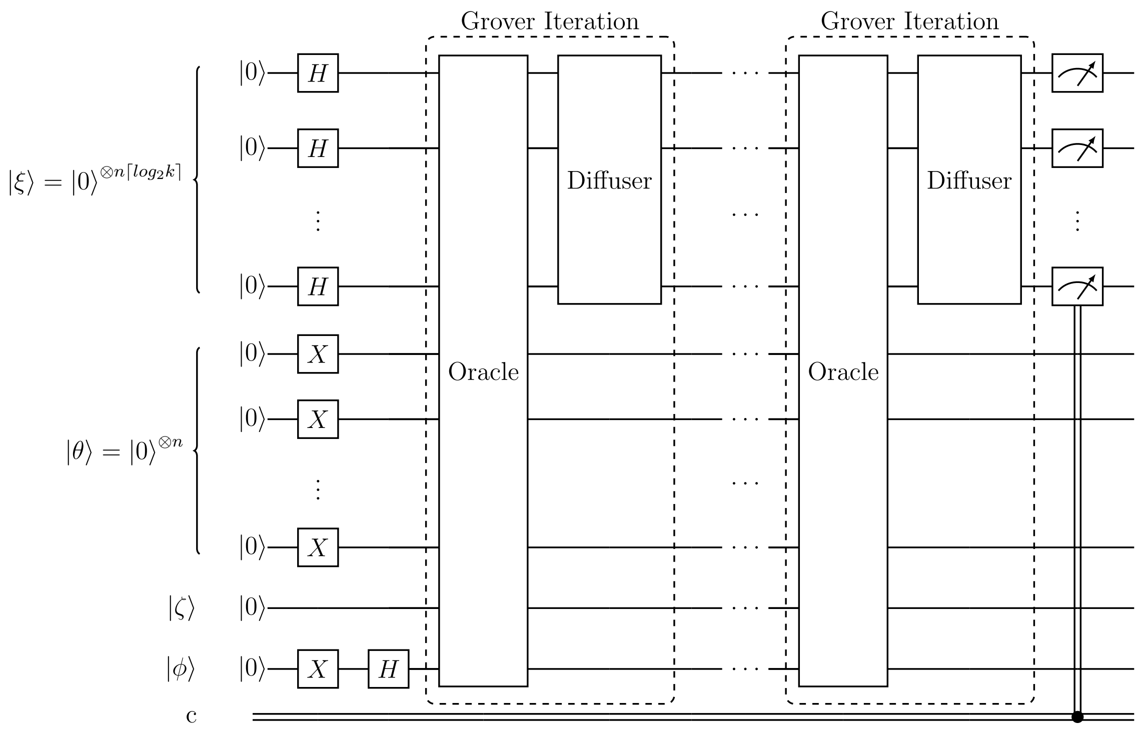

Grover Iteration
Grover Iteration
$|\xi\rangle = |0\rangle^{\otimes n\lceil log_2 k\rceil}$
$|\theta\rangle = |0\rangle^{\otimes n}$
$|\zeta\rangle$
$|\phi\rangle$
c
$|0\rangle$
H
X
Oracle
Diffuser

Figure 4.11: Full circuit for solving K-Coloring. Grover iterations are repeated $O(\sqrt{\frac{N}{M}})$ times according to equation 2.72

Chapter 5

RESULTS

In the following sections, I outline the results of algorithms run on both physical quantum computers, and simulators. Unless otherwise specified, all circuits were run 1024 times on the quantum backends. These backends are either IBMQ's *ibmq_qasm_simulator*, or *ibmq_15_melbourne*.

5.1 Results of K-Coloring

In this section, I present the results of using the quantum search algorithm to solve the K-Coloring problem for a fully connected graph with three vertices, and three colors. This graph, G_1, is shown in figure 5.1.

Using the AutoGenOracleK-Coloring algorithm, we can generate the problem specific Oracle. Once we have the Oracle, we can append the Diffuser in order to create

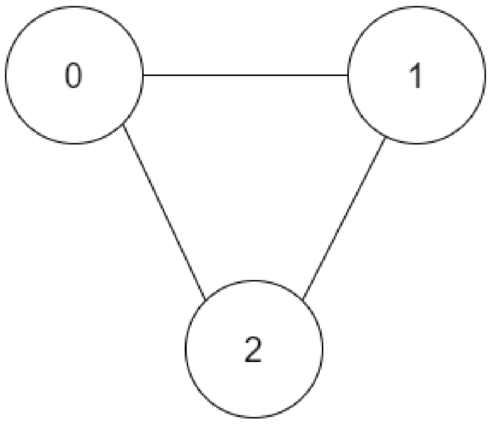


Figure 5.1: Graph G_1

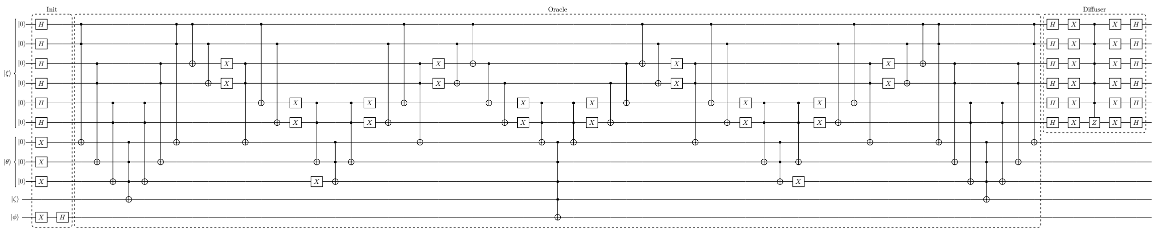


Figure 5.2: Full Grover operator for graph G_1 and $k = 3$

a circuit that implements a full Grover operator. The resulting circuit implementing this is shown in figure 5.2.

Using this circuit, we can create a counting circuit to determine the number of solutions to our Oracle. This will allow us to calculate how many times to apply the Grover operator in order to obtain a solution state from the quantum search. Using CountK-Coloring with the same graph G_1 and number of colors $k = 3$, and $t = \lceil n/2 \rceil + 3 = 5$ counting qubits, the resulting counting circuit created is shown in figure 5.3.

This circuit was then executed 1024 times on the *ibmq_qasm_simulator*, and the results are shown in figure 5.4. The two peaks in this graph are at $|01101\rangle$ and $|10011\rangle$ respectively. These correspond to values of $x_1 = 01101_2 = 13_{10}$, and $x_2 = 10011_2 = 19_{10}$. Note that the subscripts denote that the numbers are in base 2, then base 10. Plugging these into equation 2.203, we get

$$\theta_1 = \left(\frac{x}{2^t} - 1/2\right) 2\pi = \left(\frac{13}{2^5} - 1/2\right) 2\pi = -0.58905\, rad \tag{5.1}$$

$$\theta_2 = \left(\frac{x}{2^t} - 1/2\right) 2\pi = \left(\frac{19}{2^5} - 1/2\right) 2\pi = 0.58905\, rad \tag{5.2}$$

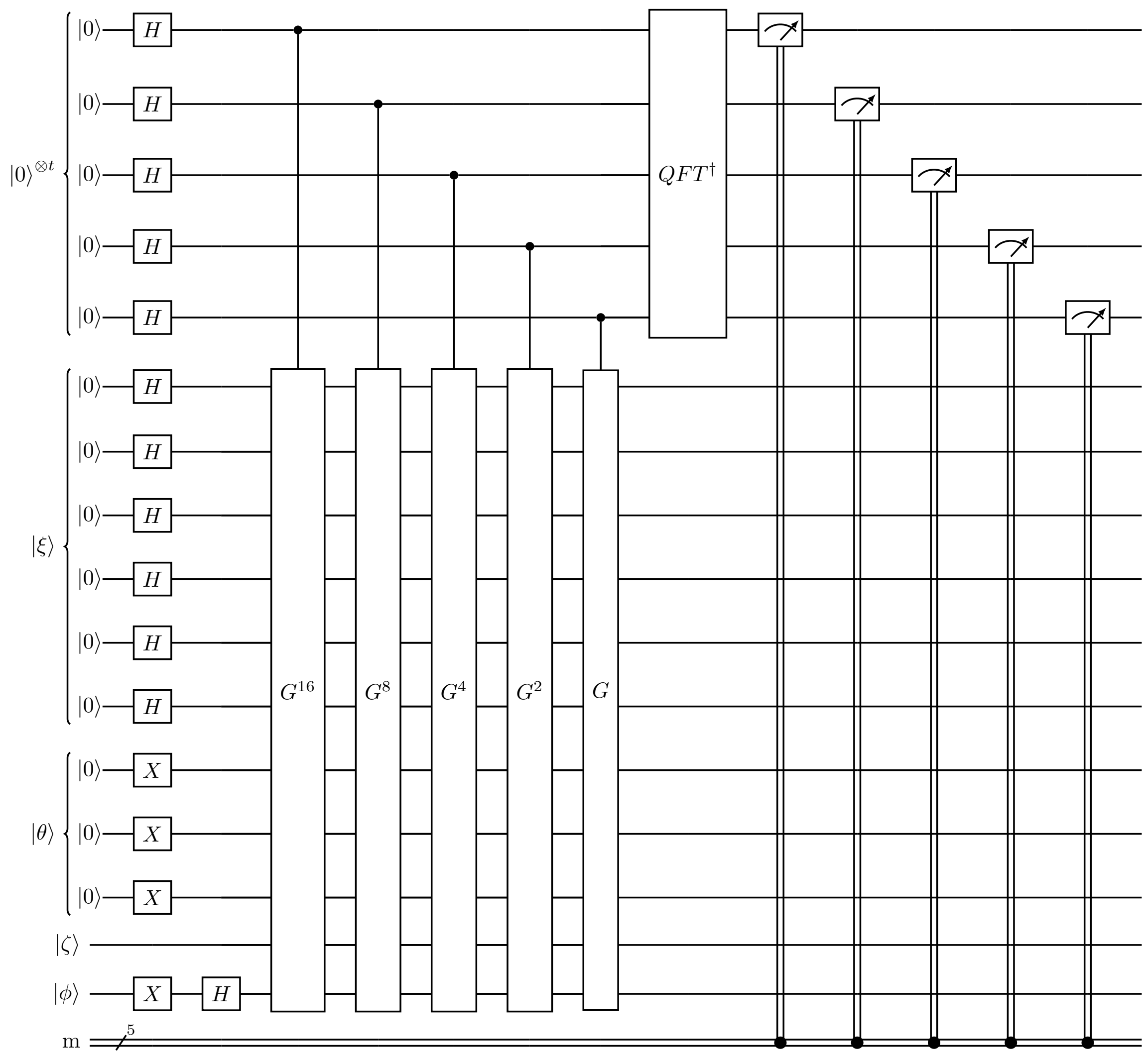


Figure 5.3: Full Quantum Counting circuit for graph G_1 and $k = 3$. Note that G represents the combined Oracle and Diffuser from circuit 5.2.

Now that we have estimates for θ, we can apply these to equation 2.76 in order to get an estimate for the number of solutions M:

$$M_1 = N \times \left(\sin\frac{\theta_1}{2}\right)^2 = 2^6 \times \left(\sin\frac{-0.58905}{2}\right)^2 = 5.4 \tag{5.3}$$

$$M_2 = N \times \left(\sin\frac{\theta_2}{2}\right)^2 = 2^6 \times \left(\sin\frac{0.58905}{2}\right)^2 = 5.4 \tag{5.4}$$

Note that $N = 2^6$ because we used 6 total qubits to represent our search space. The graph G_1 has three vertices, and we use two qubits to represent the color of each vertex. As we would expect, we see that both of these values for θ give use the same estimate for the number of solutions. Since it does not make sense to have a fraction of a solution, we will round, and take the total number of solutions to be $M = 5$.

With these estimates for M and θ, we can use equation 2.72 in order to calculate the number of times to apply the Grover:

$$R_1 = CI\left(\frac{\arccos\left(\sqrt{M/N}\right)}{\theta_1}\right) = CI\left(\frac{\arccos\left(\sqrt{5/2^6}\right)}{-0.58905}\right) = CI(-2.186) = -2 \tag{5.5}$$

$$R_2 = CI\left(\frac{\arccos\left(\sqrt{M/N}\right)}{\theta_2}\right) = CI\left(\frac{\arccos\left(\sqrt{5/2^6}\right)}{0.58905}\right) = CI(2.186) = 2 \tag{5.6}$$

We see that using the negative angle θ_1 gives us a negative amount of iterations. Since this does not make physical sense to perform negative iterations, we take the absolute value, resulting in two full iterations.

Now that we know we need to apply our Oracle and Diffuser two times in order to optimally obtain a solution state for this example, we can build the circuit that will give us a solution. This circuit is shown in figure 5.5. Here the Oracle blocks do not

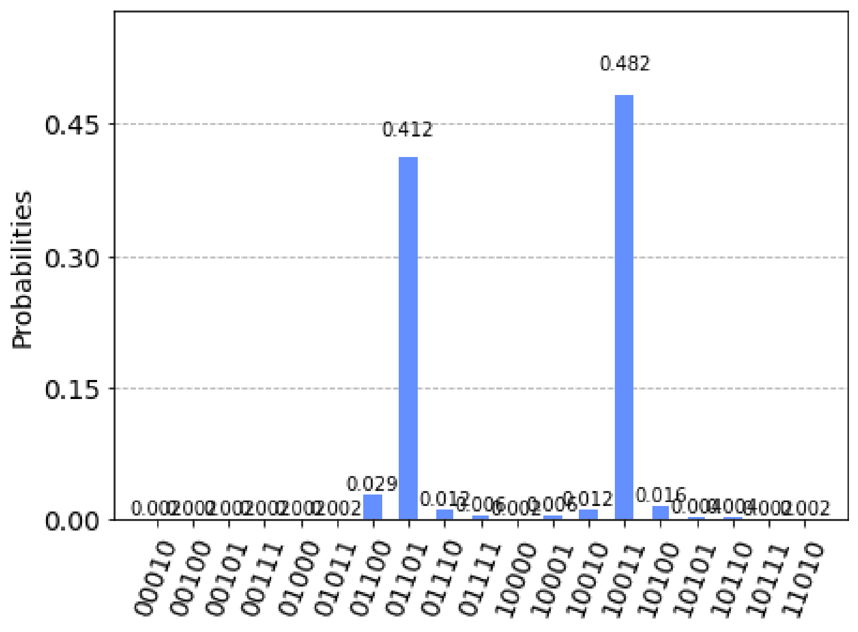


Figure 5.4: Histogram of results for counting circuit on G_1 and $k = 3$

contain the initialization steps for the qubits. This circuit was run 1024 times on *ibmq_15_melbourne*, one of the IBM Quantum Canary Processors, with a histogram of results shown in figure 5.6. As is shown in this histogram, the results of this circuit appear to be random. This is due to the individual gate errors that are present on the *ibmq_15_melbourne* chip, and to measurement errors. Approaches for addressing these errors are presented in section 6.4.

The expected solution states to this instance of the K-Coloring problem are $|000110\rangle$, $|001001\rangle$, $|100001\rangle$, $|100100\rangle$, $|010010\rangle$, and $|011000\rangle$. The color for each vertex is represented with two qubits, and these are the states that tell us that all three vertices are different colors. Note that $|11\rangle$ is an invalid color.

In order to verify that we created the correct circuit, we can run the same circuit on *ibmq_qasm_simulator*, a quantum computer simulator provided by IBM. Since this is a simulator, we can run it with no gate errors, providing a perfect output. The results of running circuit 5.5 on this simulator is shown in figure 5.7. The cir-

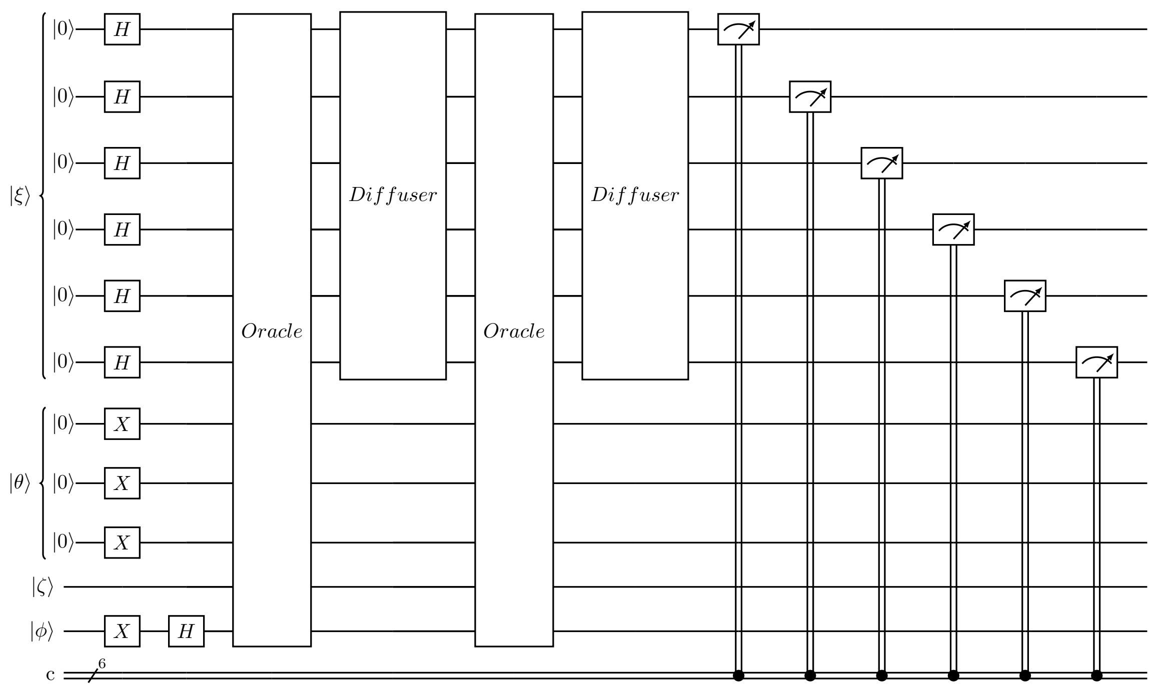


Figure 5.5: Circuit to find graph colorings for G_1 and $k = 3$

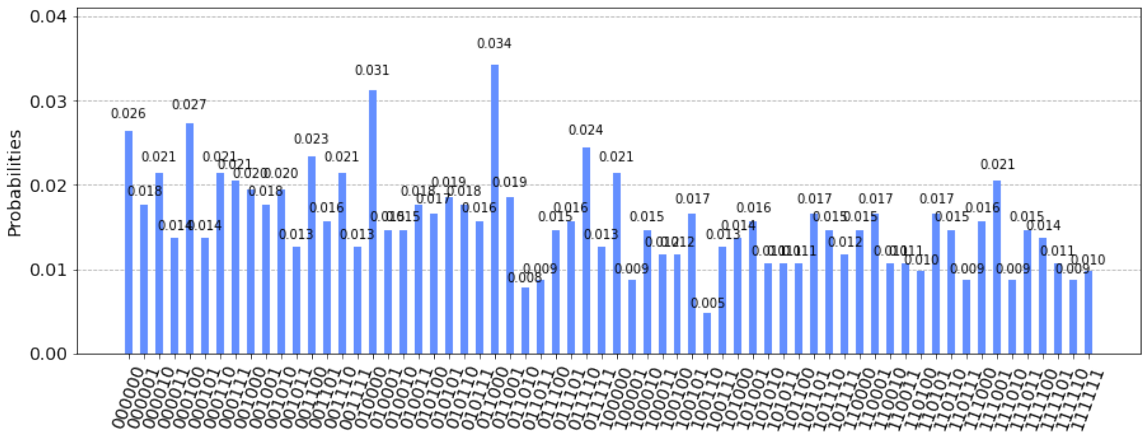


Figure 5.6: Results of running circuit 5.5 1024 times on *ibmq_15_melbourne*

cuit was run 1024 times on the simulator to obtain the histogram of results. These results show that the output of our circuit is an equal superposition of the 6 solution states for this problem. This verifies the results presented in [30] and shows that AutoGenOracleK-Coloring creates quantum circuits that can solve the K-Coloring problem. We notice that the CountK-Coloring algorithm return 5.4 as the number of solutions, whereas in actuality we have 6 solutions. This has to do with the level of precision we get with the counting qubits, and equations 2.76 and 2.203. We see that in figure 5.4 the next most common state was 01100, which corresponds to 12. Calculating M from this result we get

$$\theta = \left(\frac{x}{2^t} - 1/2\right) 2\pi = \left(\frac{12}{2^5} - 1/2\right) 2\pi = -0.78539\, rad \tag{5.7}$$

$$M = N \times \left(\sin\frac{\theta}{2}\right)^2 = 2^6 \times \left(\sin\frac{0.78539}{2}\right)^2 = 9.37 \tag{5.8}$$

We see that from this measurement, we calculated 9.37 solutions. We know that the actual number of solutions is 6, but the closest that the Quantum Counting algorithm can get is either 5.4 or 9.37. Since 5.4 is much closer to 6, Quantum Counting returned 5.4. If we wanted to increase this precision, we could add more counting qubits at the cost of run time.

5.2 Results of Chromatic Number

In this section, I present the results of the ChromaticNumber algorithm when applied to the graph G_1. This is the same graph used to evaluate the K-Coloring algorithm. As was seen in section 5.1, the errors of the *ibmq_15_melbourne* quantum system are too high to obtain reasonable results. Since this is the physical computer with the most amount of qubits that I have access to, the results in this section were

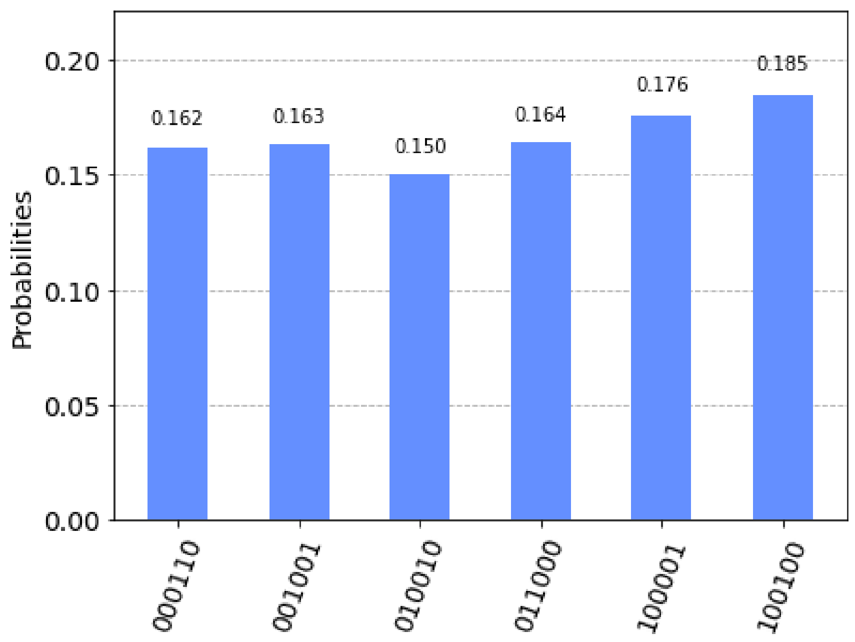


Figure 5.7: Results of running circuit 5.5 1024 times on *ibmq_qasm_simulator*. Note that all other results were measured 0 times and are omitted from the histogram.

all obtained using the *ibmq_qasm_simulator*. The other physical computers available have 5 or less qubits, whereas 5.5 uses 11 qubits.

For this problem, we start by entering the while loop in ChromaticNumber. The first time the body of the loop executes, $k_{guess} = 2$. The CountK-Coloring algorithm will generate the Oracle for graph G_1 and $k = 2$. It will then apply the Quantum Counting algorithm to this Oracle for $t = 5$. The histogram for the results of this quantum counting circuit is shown in figure 5.8. Here we can see that the result from this circuit is $x = 10000_2 = 16_{10}$. Note that the subscripts denote that the numbers are in base 2, then base 10. Following equation 2.203, we get

$$\theta = \left(\frac{x}{2^t} - 1/2\right) 2\pi = \left(\frac{16}{2^5} - 1/2\right) 2\pi = 0 \tag{5.9}$$

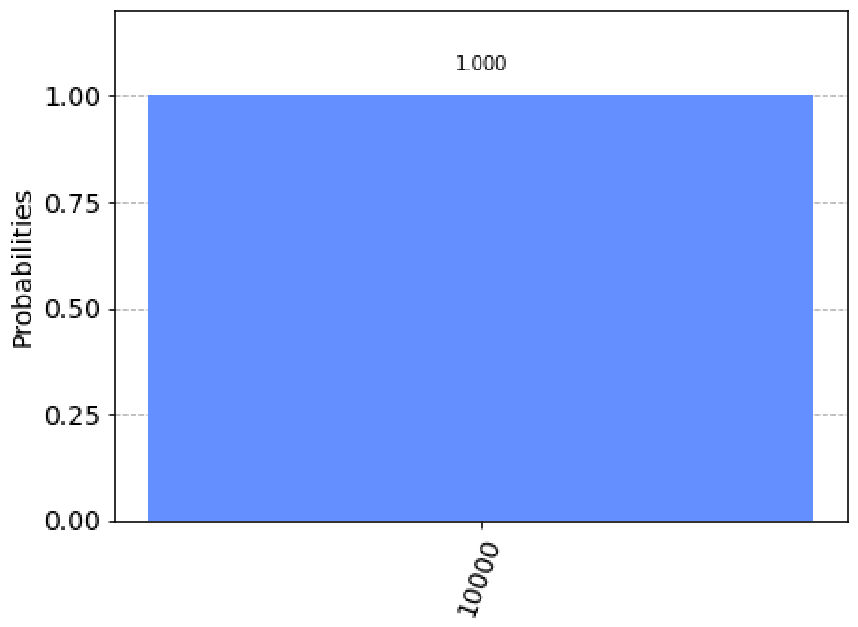


Figure 5.8: Histogram of results for counting circuit on G_1 and $k = 2$

With this value for θ, we can use equation 2.76 to calculate the number of solutions:

$$M = N \times \left(\sin \frac{\theta}{2}\right)^2 = N \times \left(\sin \frac{0}{2}\right)^2 = 0 \tag{5.10}$$

Here we can see that for $G = G_1, k = 2$ there are no solutions, which is what we would expect. Since $M = 0$, we update k_{min} to be $k_{guess} + 1 = 3$

In the next iteration of the while loop, our k_{guess} is 3. CountK-Coloring will then create and execute a counting circuit for $G = G_1, k = 3$. The results from executing this circuit are the same as in section 5.1, and are shown in figure 5.4.

Using the results from section 5.1, we see that the number of solutions is $M = 5.4$. Here we see that $M \neq 0$. This means we will update k_{max} to be $k_{guess} - 1 = 2$. When we return to the header of our while loop, we see that $k_{max} = 2$ and $k_{min} = 3$, so $k_{min} > k_{max}$. This makes the statement in our while loop false, which breaks us out of the loop. Then, since the most recent calculation for M was 5.4, we do not enter

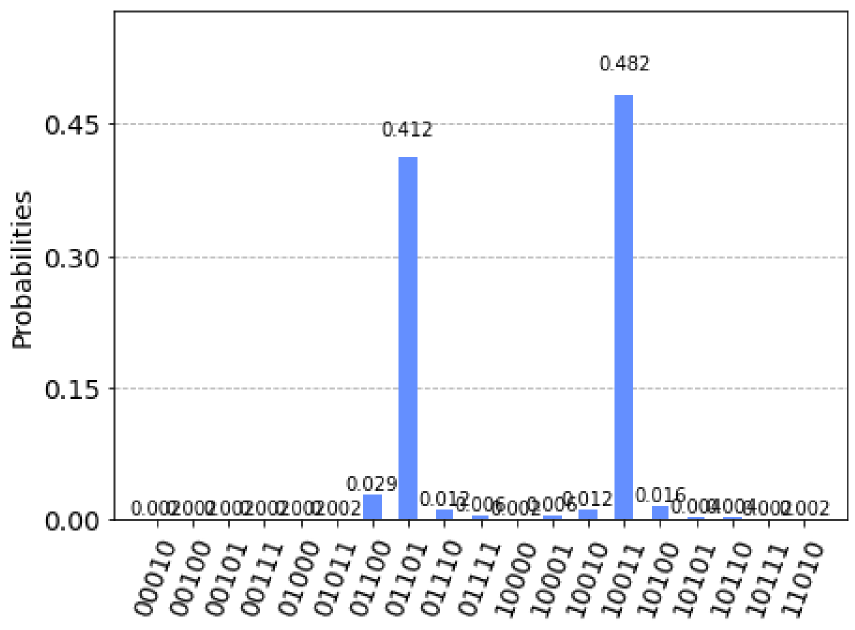


Figure 5.9: Histogram of results for counting circuit on G_1 and $k = 3$

the if statement on line 11, so we return our current value of $k_{guess} = 3$. Here we can see that the ChromaticNumber algorithm successfully calculated the chromatic number of the graph G_1.

5.3 Complexity Analysis

In this section, I analyse the time complexity of the algorithms AutoGenOracleK-Coloring, CountK-Coloring, and ChromaticNumber. I also analyze the time complexity of the Quantum Search Oracle generated by AutoGenOracleK-Coloring.

5.3.1 Complexity of K-Coloring Oracle

In order to analyze the complexity of the Oracle generated by algorithm AutoGenOracleK-Coloring, I will analyze each block shown in figure 4.8. For

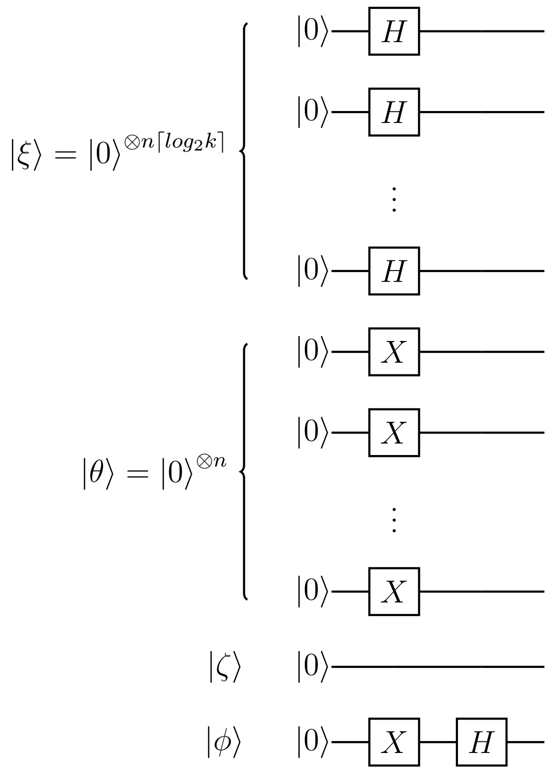

$|\xi\rangle = |0\rangle^{\otimes n\lceil log_2 k\rceil}$
$|\theta\rangle = |0\rangle^{\otimes n}$
$|\zeta\rangle$
$|\phi\rangle$
$|0\rangle$
H
X

Figure 5.10: Initialization for K-Coloring Oracle. $|\xi\rangle$ are data qubits, $|\theta\rangle$ are ancilla qubits, $|\zeta\rangle$ is invalid color qubit, and $|\phi\rangle$ is output qubit

this analysis, I will assume that each qubit starts in the state $|0\rangle$ since this is how Qiskit initializes each qubit. With this in mind, the initialization block is shown in figure 5.10. As we can see, the depth of this circuit is 2, and will stay 2 no matter how big our input n is. Thus the time complexity of the initialization is $O(1)$.

The next block we will evaluate is the Invalid Color Detector. The general form that this block will take is shown in figure 4.2. Recall that each partial Invalid Color Detector takes the form of the circuit in 4.3. Here we see that each MCT has $\lceil log_2 k \rceil$ controls, since that is how many qubits we need to represent all k colors. Additionally, we see that we sometimes apply X-gates before and after the MCT, but since the depth of those X-gates is constant with respect to the number of qubits, they have time complexity $O(1)$. We need to add a partial Invalid Color Detector for each invalid color, of which there will be at most 129 colors. In this case we have $\lceil log_2 129 \rceil = 8$ qubits. The number of invalid colors in this case is $2^8 - 129 = 127$. In the worst case, $k = n$. Thus, to calculate the complexity of circuit 4.2 up to slice 2, we do the following:

$$(\#\text{ of vertices}) \times [(\#\text{ invalid colors}) \times (\text{Complexity of } \lceil log_2 k \rceil\text{-MCT gate})] \tag{5.11}$$

Since we have n vertices, and in the worst case n invalid colors, with each color represented by $\lceil log_2 n \rceil$ qubits, we get:

$$(n) \times [(n) \times (log_2 n)] = O(n^2 log_2 n) \tag{5.12}$$

This uses the results of [22] to get the complexity of the MCT gate.

After slice 2 in figure 4.2, we apply an MCT controlled by n ancilla qubits, which gives us complexity $O(n)$. Finally, the un-computation is the same as the complexity up until slice 2, $O(n^2 log_2 n)$. Therefore the overall complexity of the Invalid Color

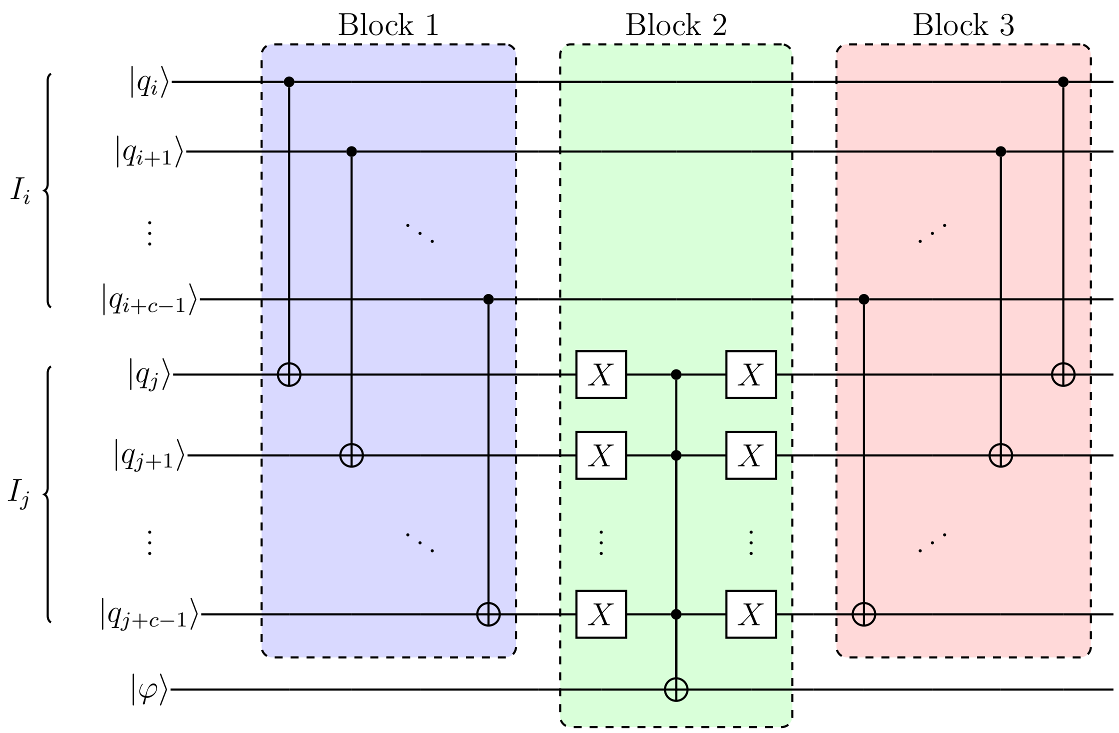


Figure 5.11: Comparator circuit for two colors. Note that in implementation $|\varphi\rangle$ will be one of the ancilla qubits $|\theta_i\rangle$

Detector is

$$O(\text{ICD}) = O(O(n^2 log_2 n) + O(n) + O(n^2 log_2 n)) = O(n^2 log_2 n) \tag{5.13}$$

Next we analyze the complexity of the comparator block. Recall from algorithm AutoGenOracleK-Coloring that we add a comparator to the circuit for *each* edge in the graph $G = (V, E)$. In the worst case we will have $n(n-1)$ edges, where n is the number of vertices. Thus we can see that the overall complexity of the comparator block will be:

$$(\#\text{ of edges}) \times (O(\text{single Comparator})) = n(n-1) \times (O(\text{single Comparator})) \tag{5.14}$$

In order to analyze the complexity of a single comparator, we will look at circuit 5.11. We see that in block 1, we apply a cNOT gate for each qubit used to represent the color of a single vertex. In the worst case, this will be $\lceil log_2 n \rceil$ qubits, resulting in a total of $\lceil log_2 n \rceil$ cNOT gates. Since the complexity of each cNOT gate is $O(1)$, the complexity of block 1 is $O(log_2 n)$. In block 2, the complexity of the X-gates is $O(1)$ since they can all be executed in parallel. We also have a MCT gate controlled by $\lceil log_2 n \rceil$ qubits, which from [22], we know has a complexity of $O(log_2 n)$. Finally the block 3 is the reverse of block 1, resulting in a complexity of $O(log_2 n)$. So the overall complexity of a single comparator is:

$$O(O(\text{Block 1}) + O(\text{Block 2}) + O(\text{Block 3})) = O(log_2 n + log_2 n + log_2 n) = O(log_2 n) \tag{5.15}$$

So returning to the complexity of the overall comparator block, we can see that it will be

$$O(n((n-1) \times O(\text{single Comparator}))) = O(n^2 log_2 n) \tag{5.16}$$

The next block in circuit 4.8 is a single MCT gate controlled by $n+1$ qubits, which will have complexity $O(n)$.

The second comparator block is the same as the first block, and so will have the same complexity. The same applies for the second Invalid Color Detector block. Finally, we have a Diffuser, which can be implemented as 2.24. In this implementation, the depth of the H and X gates is constant with respect to the number of qubits. Then the n-MCZ gate has time complexity $O(n)$. Thus the overall complexity of the Diffuser is $O(n)$.

Putting this all together, we get:

$$\begin{aligned} O(\text{Oracle}) = O\Big(&O(\text{Init}) + O(\text{ICD}) + O(\text{Comparator Block}) + O(n\text{-MCT}) \\ &+ O(\text{Comparator Block}) + O(\text{ICD}) + O(\text{Diffuser})\Big) \\ &= O(1 + n^2 log_2 n + n^2 log_2 n + n + n^2 log_2 n + n^2 log_2 n + n) \\ &= O(4n^2 log_2 n + 2n + 1) \\ &= O(n^2 log_2 n) \end{aligned} \tag{5.17}$$

5.3.2 Complexity of AutoGenOracleK-Coloring

For this algorithm, the most computationally intensive component actually comes from applying the Invalid Color Detector to the circuit. The pseudo code for the Invalid Color Detector is given below:

```
1: for each vertex $v_i \in V$ do
2:     for each invalid color $c$ do
3:         Add X-gates and MCT such that output $|\phi\rangle = 1$ if $|\text{data qubits}\rangle = |c\rangle$,
   and $|\phi\rangle = 0$ else.
```

The number of vertices in the graph is $|V| = n$. In the worst case, the number of colors we need is $k = n$, needing $\lceil log_2 n \rceil$ qubits to represent. The number of invalid colors we have is given by $2^{\lceil log_2 k \rceil} - k$, which in the worst case is equal to k, $2^{\lceil log_2 k \rceil} - k \approx k$. Thus we can assume that the loop on line 2 executes n times. Finally, the implementation of line 3 takes $log_2 k$ time, since we need to do a constant number of operations per data qubit, where we have $log_2 k$ data qubits. Thus all put together we have

$$O(n((2^{\lceil log_2 k\rceil} - k)(log_2 k))) = O(n((2^{\lceil log_2 n\rceil} - n)(log_2 n))) = O(n(n(log_2 n))) \\ = O(n^2 log_2 n) \tag{5.18}$$

Given this, the overall complexity of AutoGenOracleK-Coloring is $O(n^2 log_2 n)$.

5.3.3 Complexity of CountK-Coloring

For this algorithm, the complexity comes from running the Quantum Counting algorithm itself. The final circuit that we build will have a total of 2^t K-Coloring Oracles. Thus when we run the circuit the complexity will be $O(2^t \times \{\text{Complexity Of Oracle}\})$. As previously discussed, the value of t will be set to $\lceil n/2\rceil + 3$, so the overall complexity is

$$O(\sqrt{2^n} \times \{\text{Complexity Of Oracle}\}) = O(\sqrt{2^n} n^2 log_2 n) \tag{5.19}$$

5.3.4 Complexity of ChromaticNumber

Since we are implementing binary search to find the value k which is our chromatic number, we know that the search loop will run $O(log_2 n)$ times. In each search iteration, we call CountK-Coloring. Thus the overall complexity is

$$O(log_2 n \times \{\text{Complexity of CountK-Coloring}\}) = O(\sqrt{2^n} n^2 (log_2 n)^2) \tag{5.20}$$

Here, we see that we can solve the Chromatic Number problem in
time $O(\sqrt{2^n} n^2 (log_2 n)^2)$.

We have thus successfully implemented the ChromaticNumber algorithm on a quantum computer, and found its running time to be $O(\sqrt{2^n}n^2(log_2 n)^2)$. This is an improvement over the fastest classical algorithm which runs in time $2^n n^{O(1)}$ [1].

Chapter 6

FUTURE WORK

In this section, I outline possible future additions to the work presented in this thesis.

6.1 General Framework for NP-Complete Problems

As discussed in section 3, there has already been work that proposed general frameworks to solve NP-Complete problems using quantum hardware [36, 33]. The frameworks presented in these works both focus on reducing NP-Complete problems to the Satisfiability (SAT) problem, then solving the SAT problem using Grover's algorithm. It is not important to understand the details of the SAT problem, only that it is one of many NP-Complete problems. The work serves mainly as a proof of concept, and has room for improvements. One of the drawbacks of this approach is that the reduction can create more resource intensive circuits than necessarily required to solve the problem. The authors of [33] present an Oracle to solve the 3-coloring problem (a simplification of the K-Coloring problem discussed in this thesis) using their reduction based technique, which ends up using many more resources than the approach presented here. As analyzed by [30], the Oracle in [33] uses $n \times k$ data qubits and $O((n \times k)^2)$ ancillary qubits, whereas the Oracle used here uses $n \times \lceil log_2 k \rceil$ data qubits, and $O(n)$ ancillary qubits. With this in mind, I propose that a general framework for solving NP-Complete problems use as many specific Oracles as available in order to provide the fastest and least resource intensive results possible. For example the framework could include the Oracle for the SAT problem as well as the K-Coloring.

Then for some new problem, it could reduce based on heuristics to either of the known Oracles.

6.2 Optimized Quantum Counting

The basis of the ChromaticNumber algorithm is the Quantum Counting algorithm. Additionally the majority of the time complexity of this algorithm comes from the repeated applications of the Quantum Search Oracle by the Quantum Counting algorithm. The number of applications is $\sqrt{2^n}$ which gives us the complexity of $O(\sqrt{2^n}n^2(log_2 n)^2)$. If we can reduce the number of times we need to apply the Oracle, we can reduce the overall complexity of ChromaticNumber. In [37], Wie presents a different approach to the Quantum Counting algorithm; Simpler Quantum Counting. The time complexity is the same, but it is experimentally faster than regular Quantum Counting. Additionally, the Simpler Quantum Counting uses only 1 additional qubit instead of the t qubits required by regular Quantum Counting. Wie experimentally shows that for small values of M/N, the simpler algorithm performs faster than the original Quantum Counting algorithm. Application of the Simpler Quantum Counting algorithm to the solution of ChromaticNumber will not improve the asymptotic time complexity, but may perform better in practice. Additionally it will reduce the number of qubits required to run the algorithm, which allows for sooner real world application.

6.3 Exact algorithm for Grover Search

The Quantum Search algorithm is not a deterministic algorithm. The probability of success can range from one half to 1 depending on the values of N and M [16]. Because of this, in order to be sure of a solution returned by the Quantum Search algorithm, the algorithm must be run more than once. This will slow down the solution of a given problem. In [38], Long presents a modified version of the Quantum Search algorithm that has zero theoretical failure rate. This means that after running the algorithm, there is a 100% chance that the state measured is the actual solution state. Incorporating this technique into the solution of K-Coloring would improve the practical run time of the algorithm.

6.4 Error Correction and Fault Tolerance

As mentioned in the Results section, the *ibmq_15_melbourne* had too much inherent error to successfully run the Oracle and Counting circuits developed here. Noise in quantum computers has been a well studied and well known issue in the field for over 20 years. As such, a variety of techniques have been developed to help mitigate the errors caused by this noise [11, 39]. Applications of these techniques to the algorithms outlined in this paper would improve the practicality of these algorithms, and make them more feasible in the short term. However it is worth noting that many of these techniques require additional qubits, which make them less applicable in short term. It will be important to find a balance between the amount of error correction, and the amount of actual algorithmic processing.

6.5 Analyze Time Per Basic Operation

Another interesting route for this work, is to analyze when the actual solution times of quantum computers overtake those of classical computers. We can see from the complexity analysis that the theoretical time for solution of the Chromatic Number problem is faster on a quantum computer than a classical computer. However the question that this begs, is when it is actually practical to use quantum computers to solve these types of problems. As of writing this, there are almost no real world tasks that actually benefit from running on quantum computers. This is due to errors in the computers, setup time for circuits, limited access which corresponds to high wait times for a job to be run, and limited number of qubits. So when is the actual computation faster on quantum hardware? Comparing the individual gate times on quantum hardware with the individual instruction times for classical computers could provide some answers to this question.

Chapter 7

CONCLUSION

In this work, I presented the algorithm ChromaticNumber which solves the Chromatic Number problem. I then analyzed the complexity of this algorithm and found it to run in time $O(\sqrt{2^n}n^2(log_2 n)^2)$. This is an improvement over the best classical algorithms which solve the problem in time $2^n n^{O(1)}$ [1]. Additionally, I provided a slightly modified quantum Oracle for the K-Coloring problem, taken from [30]. While the physical quantum computer suffered from too much noise to accurately run the K-Coloring Oracle, or the ChromaticNumber algorithm, the results from IBM's *ibmq_qasm_simulator* suggest that these algorithms do indeed solve their respective problems on quantum devices. While the speed up offered by these algorithms is not enormous, it is also not insignificant. As classical computation reaches the end of Moore's law, any potential for improved efficiency is significant. Additionally, as quantum technology continues to improve, the adoption of these technologies to solve real world problems will increase. As such, it is important for traditional computer scientists to develop an understanding of these quantum technologies so they can be best utilized moving forward.

BIBLIOGRAPHY

[1] A. Björklund, T. Husfeldt, and M. Koivisto, "Set Partitioning via Inclusion-Exclusion," *SIAM Journal on Computing*, vol. 39, pp. 546–563, Jan. 2009. Publisher: Society for Industrial and Applied Mathematics.

[2] C. Qiu, Z. Zhang, M. Xiao, Y. Yang, D. Zhong, and L.-M. Peng, "Scaling carbon nanotube complementary transistors to 5-nm gate lengths," *Science*, vol. 355, pp. 271–276, Jan. 2017. Publisher: American Association for the Advancement of Science Section: Report.

[3] A. S. Holevo, "Bounds for the quantity of information transmitted by a quantum communication channel," *Probl. Peredachi Inf.*, vol. 9, no. 3, pp. 3–11, 1973.

[4] R. S. Ingarden, "Quantum information theory," *Reports on Mathematical Physics*, vol. 10, pp. 43–72, Aug. 1976.

[5] P. Benioff, "The computer as a physical system: A microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines," *Journal of Statistical Physics*, vol. 22, pp. 563–591, May 1980.

[6] R. Feynman, "Simulating physics with computers," *International Journal of Theoretical Physics*, vol. 21, 1982.

[7] P. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," in *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pp. 124–134, Nov. 1994.

[8] E. W. Weisstein, "Number Field Sieve." Publisher: Wolfram Research, Inc. https://mathworld.wolfram.com/NumberFieldSieve.html Accessed: 2021-05-28.

[9] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proceedings of the twenty-eighth annual ACM symposium on Theory of Computing*, STOC '96, (New York, NY, USA), pp. 212–219, Association for Computing Machinery, July 1996.

[10] R. M. Karp, "Reducibility among Combinatorial Problems," in *Complexity of Computer Computations: Proceedings of a symposium on the Complexity of Computer Computations, held March 20–22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, and sponsored by the Office of Naval Research, Mathematics Program, IBM World Trade Corporation, and the IBM Research Mathematical Sciences Department* (R. E. Miller, J. W. Thatcher, and J. D. Bohlinger, eds.), The IBM Research Symposia Series, pp. 85–103, Boston, MA: Springer US, 1972.

[11] M. A. Nielsen and I. L. Chuang, *Quantum computation and quantum information*. Cambridge ; New York: Cambridge University Press, 10th anniversary ed ed., 2010.

[12] H. Abraham, AduOffei, and R. A. et. al., "Qiskit: An open-source framework for quantum computing," 2019.

[13] A. Kay, "Tutorial on the Quantikz Package,"

[14] M. A. Nielsen and I. L. Chuang, *Quantum computation and quantum information*, pp. 177–185. Cambridge ; New York: Cambridge University Press, 10th anniversary ed ed., 2010.

[15] G. L. Long, Y. S. Li, W. L. Zhang, and L. Niu, “Phase matching in quantum searching,” *Physics Letters A*, vol. 262, pp. 27–34, Oct. 1999.

[16] M. A. Nielsen and I. L. Chuang, *Quantum computation and quantum information*, pp. 248–263. Cambridge ; New York: Cambridge University Press, 10th anniversary ed ed., 2010.

[17] S. Roberts, “The discrete fourier transform,” 2017. https://www.robots.ox.ac.uk/~sjrob/Teaching/SP/l7.pdf.

[18] O. A. Ivanova, “Geometric progression.” http://encyclopediaofmath.org/index.php?title=Geometric_progression&oldid=12512 Accessed: 2021-05-28.

[19] G. Brassard, P. HØyer, and A. Tapp, “Quantum counting,” in *Automata, Languages and Programming* (K. G. Larsen, S. Skyum, and G. Winskel, eds.), Lecture Notes in Computer Science, (Berlin, Heidelberg), pp. 820–831, Springer, 1998.

[20] A. Mohr, “Quantum Computing in Complexity Theory and Theory of Computation,” p. 6.

[21] M. A. Nielsen and I. L. Chuang, *Quantum computation and quantum information*, p. 212. Cambridge ; New York: Cambridge University Press, 10th anniversary ed ed., 2010.

[22] L. Biswal, D. Bhattacharjee, A. Chattopadhyay, and H. Rahaman, “New techniques for fault-tolerant decomposition of Multi-Controlled Toffoli gate,” *Physical Review A*, vol. 100, p. 062326, Dec. 2019. arXiv: 1904.06920.

[23] N. J. Cerf, L. K. Grover, and C. P. Williams, “Nested quantum search and NP-complete problems,” *Physical Review A*, vol. 61, p. 032303, Feb. 2000. arXiv: quant-ph/9806078.

[24] G. Hao, L. Gui-Lu, and L. Feng, “Quantum Algorithms for Some Well-Known NP Problems,” *Communications in Theoretical Physics*, vol. 37, pp. 424–426, Apr. 2002. Publisher: IOP Publishing.

[25] M. Fürer, “Solving NP-Complete Problems with Quantum Search,” in *LATIN 2008: Theoretical Informatics* (E. S. Laber, C. Bornstein, L. T. Nogueira, and L. Faria, eds.), Lecture Notes in Computer Science, (Berlin, Heidelberg), pp. 784–792, Springer, 2008.

[26] L. Wen-Zhang, Z. Jing-Fu, and L. Gui-Lu, “A Parallel Quantum Algorithm for the Satisfiability Problem,” *Communications in Theoretical Physics*, vol. 49, pp. 629–630, Mar. 2008. Publisher: IOP Publishing.

[27] W. Chang, T. Ren, M. Feng, S. Huang, L. C. Lu, K. W. Lin, and M. Guo, “Quantum Algorithms of the Vertex Cover Problem on a Quantum Computer,” in *2009 WASE International Conference on Information Engineering*, vol. 2, pp. 100–103, July 2009.

[28] C. R. Wie, “A Quantum Circuit to Construct All Maximal Cliques Using Grover Search Algorithm,” *arXiv:1711.06146 [quant-ph]*, July 2019. arXiv: 1711.06146.

[29] A. Sanyal, A. Saha, B. Saha, and A. Chakrabarti, “Circuit Design Of Clique Problem And Its Implementation On NISQ Using Combinatorial Approach Of Classical-Quantum Hybrid Model,” *arXiv:2004.10596 [cs]*, Jan. 2021. arXiv: 2004.10596.

[30] A. Saha, D. Saha, and A. Chakrabarti, “Circuit Design for K-coloring Problem and it’s Implementation on Near-term Quantum Devices,” *arXiv:2009.06073 [cs]*, Sept. 2020. arXiv: 2009.06073.

[31] A. Saha, A. Chongder, S. B. Mandal, and A. Chakrabarti, “Synthesis of Vertex Coloring Problem Using Grover’s Algorithm,” in *2015 IEEE International Symposium on Nanoelectronic and Information Systems*, pp. 101–106, Dec. 2015.

[32] B. Moon, “The Subset Sum Problem: Reducing Time Complexity of NP-Completeness with Quantum Search,” *Undergraduate Journal of Mathematical Modeling: One Two*, vol. 4, Jan. 2012.

[33] S. Hu, P. Liu, C. R. Chen, M. Pistoia, and J. Gambetta, “Reduction-Based Problem Mapping for Quantum Computing,” *Computer*, vol. 52, pp. 47–57, June 2019. Conference Name: Computer.

[34] E. Farhi, J. Goldstone, and S. Gutmann, “A Quantum Approximate Optimization Algorithm,” *arXiv:1411.4028 [quant-ph]*, Nov. 2014. arXiv: 1411.4028.

[35] S. Pemmaraju and S. Skiena, *Computational Discrete Mathematics: Combinatorics and Graph Theory with Mathematica ®*. Cambridge University Press, 2009.

[36] S. Hu, P. Liu, C.-F. Chen, and M. Pistoia, “Poster: Automatically Solving NP-Complete Problems on a Quantum Computer,” in *2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion)*, pp. 258–259, May 2018. ISSN: 2574-1934.

[37] C. R. Wie, “Simpler Quantum Counting,” *Quantum Information and Computation*, vol. 19, no. 11&12, 2019. arXiv: 1907.08119.

[38] G. L. Long, “Grover algorithm with zero theoretical failure rate,” *Physical Review A*, vol. 64, p. 022307, July 2001. Publisher: American Physical Society.

[39] S. J. Devitt, K. Nemoto, and W. J. Munro, “Quantum Error Correction for Beginners,” *Reports on Progress in Physics*, vol. 76, p. 076001, July 2013. arXiv: 0905.2794.

[40] “Cal Poly Github.” http://www.github.com/CalPoly.

APPENDICES

Appendix A

QUANTUM GATE REFERENCES

Table A.1: Quantum Gates

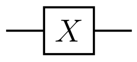
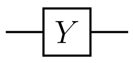
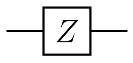
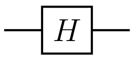
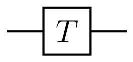
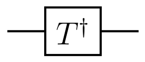
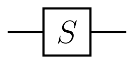
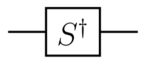
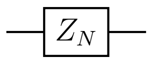
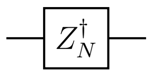
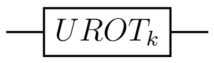
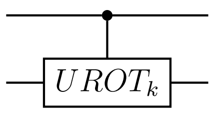
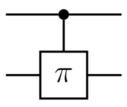
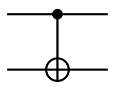
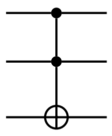
Quantum Gate	Matrix	Circuit	Operations
X-gate	$\begin{bmatrix}0 & 1\\1 & 0\end{bmatrix}$		$X\|0\rangle = \|1\rangle$ $X\|1\rangle = \|0\rangle$
Y-gate	$\begin{bmatrix}0 & -i\\i & 0\end{bmatrix}$		$Y\|0\rangle = i\|1\rangle$ $Y\|1\rangle = -i\|0\rangle$
Z-gate	$\begin{bmatrix}1 & 0\\0 & -1\end{bmatrix}$		$Z\|0\rangle = \|0\rangle$ $Z\|1\rangle = -\|1\rangle$
Hadamard (H)-gate	$\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\1 & -1\end{bmatrix}$		$H\|0\rangle = \frac{1}{\sqrt{2}}(\|0\rangle + \|1\rangle)$ $H\|1\rangle = \frac{1}{\sqrt{2}}(\|0\rangle - \|1\rangle)$
T-gate	$\begin{bmatrix}1 & 0\\0 & e^{i\pi/4}\end{bmatrix}$		$T\|0\rangle = \|0\rangle$ $T\|1\rangle = e^{\frac{i\pi}{4}}\|1\rangle$
T†-gate	$\begin{bmatrix}1 & 0\\0 & e^{-i\pi/4}\end{bmatrix}$		$T^\dagger\|0\rangle = \|0\rangle$ $T^\dagger\|1\rangle = e^{-\frac{i\pi}{4}}\|1\rangle$
S-gate	$\begin{bmatrix}1 & 0\\0 & e^{i\pi/2}\end{bmatrix}$		$S\|0\rangle = \|0\rangle$ $S\|1\rangle = e^{\frac{i\pi}{2}}\|1\rangle$
S†-gate	$\begin{bmatrix}1 & 0\\0 & e^{-i\pi/2}\end{bmatrix}$		$S^\dagger\|0\rangle = \|0\rangle$ $S^\dagger\|1\rangle = e^{\frac{-i\pi}{2}}\|1\rangle$
Z$_N$-gate	$\begin{bmatrix}1 & 0\\0 & e^{\frac{i\pi}{N}}\end{bmatrix}$		$Z_N\|0\rangle = \|0\rangle$ $Z_N\|1\rangle = e^{\frac{i\pi}{N}}\|1\rangle$
Z$_N^\dagger$-gate	$\begin{bmatrix}1 & 0\\0 & e^{-\frac{i\pi}{N}}\end{bmatrix}$		$Z_N^\dagger\|0\rangle = \|0\rangle$ $Z_N\dagger\|1\rangle = e^{-\frac{i\pi}{N}}\|1\rangle$
UROT$_k$-gate	$\begin{bmatrix}1 & 0\\0 & e^{\frac{2\pi i}{2^k}}\end{bmatrix}$		$UROT_k\|0\rangle = \|0\rangle$ $UROT_k\|1\rangle = e^{\frac{2\pi i}{2^k}}\|1\rangle$
CROT$_k$-gate	$\begin{bmatrix}1 & 0 & 0 & 0\\0 & 1 & 0 & 0\\0 & 0 & 1 & 0\\0 & 0 & 0 & e^{\frac{2\pi i}{2^k}}\end{bmatrix}$		$\|00\rangle \to \|00\rangle$ $\|01\rangle \to \|01\rangle$ $\|10\rangle \to \|10\rangle$ $\|11\rangle \to e^{\frac{2\pi i}{2^k}}\|11\rangle$
cPHASE-gate	$\begin{bmatrix}1 & 0 & 0 & 0\\0 & 1 & 0 & 0\\0 & 0 & 1 & 0\\0 & 0 & 0 & e^{i\phi}\end{bmatrix}$		$\|00\rangle \to \|00\rangle$ $\|01\rangle \to \|01\rangle$ $\|10\rangle \to \|10\rangle$ $\|11\rangle \to e^{i\phi}\|11\rangle$
cNOT-gate	$\begin{bmatrix}1 & 0 & 0 & 0\\0 & 1 & 0 & 0\\0 & 0 & 0 & 1\\0 & 0 & 1 & 0\end{bmatrix}$		$\|00\rangle \to \|00\rangle$ $\|01\rangle \to \|01\rangle$ $\|10\rangle \to \|11\rangle$ $\|11\rangle \to \|10\rangle$

Table A.2: Quantum Gates Continued

Quantum Gate	Matrix	Circuit	Operations
Toffoli-gate	$\begin{bmatrix}1&0&0&0&0&0&0&0\\0&1&0&0&0&0&0&0\\0&0&1&0&0&0&0&0\\0&0&0&1&0&0&0&0\\0&0&0&0&1&0&0&0\\0&0&0&0&0&1&0&0\\0&0&0&0&0&0&0&1\\0&0&0&0&0&0&1&0\end{bmatrix}$		$\lvert 000\rangle \rightarrow \lvert 000\rangle$ $\lvert 001\rangle \rightarrow \lvert 001\rangle$ $\lvert 010\rangle \rightarrow \lvert 011\rangle$ $\lvert 011\rangle \rightarrow \lvert 010\rangle$ $\lvert 100\rangle \rightarrow \lvert 100\rangle$ $\lvert 101\rangle \rightarrow \lvert 101\rangle$ $\lvert 110\rangle \rightarrow \lvert 111\rangle$ $\lvert 111\rangle \rightarrow \lvert 110\rangle$