Noisy Quantum Oracles: A Study of Algorithmic Robustness

A Thesis
Presented to
The Division of Mathematics and Natural Sciences
Reed College

In Partial Fulfillment
of the Requirements for the Degree
Bachelor of Arts

Marika Swanberg

May 2019

Approved for the Division
(Mathematics)

_____

James Pommersheim

Preface

"It's not a Turing machine, but a machine of a different kind" [Feynman, 1982]. Quantum computing was proposed by Richard Feynman in 1981 for simulating quantum physical phenomena. He was the first to suggest that a computer based upon quantum mechanics could potentially have a computational advantage to classical computers.

This idea became relevant to mainstream computer science theorists in 1994 when Peter Shor challenged the Extended Church-Turing thesis[1] which they held so dearly. He proposed a quantum algorithm that could compute the prime factors of integers in polynomial time, a large speedup compared to classical computers [Shor, 1999]. This algorithm demonstrated the great potential of quantum computation and spurred on the research community.

Although Shor's algorithm was provocative *in theory*, actual implementations of quantum computers lagged behind. In 2001, scientists published their success at physically implementing Shor's algorithm to compute the prime factors of the number 15 [Vandersypen et al., 2001]. Clearly, there was a large gap to bridge between theory and practice.

Noise and inaccuracy are still among the largest obstacles to making quantum computation feasible. Quantum physical systems are plagued by quantum decoherence, the tendency of particles to be influenced by their environment and thus be difficult to preserve properly. In this thesis, we study error-correcting codes and the robustness of quantum algorithms to noise. Managing errors and noise will be key to realizing the full potential of quantum computing.

[1]The Extended Church-Turing thesis (previously called the quantitative Church-Turing thesis) is stated by Shor as follows: *any physical computing device can be simulated by a Turing machine in a number of steps polynomial in the resources used by the computing device* [Shor, 1999]. In most cases, "resources" refers to either time (CPU cycles) or space (memory).

Contents

Chapter 1: Quantum Computing **1**
1.1 Information Representation 2
1.2 Quantum Logic Gates 3
1.3 Multiple Qubits 4
1.4 Entangled States 5
1.5 Hadamard Transform 6
1.6 Measuring in Other Bases 7
1.7 Phase 7
1.8 Quantum Oracles 8
1.8.1 Quantum Queries 8
1.9 Phase-kickback Trick 9
1.10 Bernstein-Vazirani Algorithm 9
1.11 Advanced Topics 11
1.11.1 No-Cloning Theorem 11
1.11.2 Universal Quantum Gate 12

Chapter 2: Error-Correcting Codes **13**
2.1 Preliminaries 13
2.2 Overview of Transmission 14
2.3 Encoding 14
2.4 Decoding 16
2.4.1 Properties of Vectors 16
2.4.2 Errors 17
2.4.3 Properties of Codes 18
2.4.4 Decoding Errors 18

Chapter 3: Learning Theory **21**
3.1 Basics of Concept Learning 21

Chapter 4: The Binary Simplex Code **23**

4.1 Encoding . . . 23
4.1.1 Generating Matrix . . . 24
4.2 Quantum Decoding Algorithm . . . 24
4.2.1 Reliable Quantum Oracle . . . 25
4.2.2 Algorithm . . . 25
4.3 Rosbustness to Errors . . . 27
4.3.1 Noisy Quantum Oracle . . . 27
4.3.2 Algorithm with Noisy Oracle . . . 27

Chapter 5: Reed-Muller Codes . . . 31
5.1 Boolean Functions . . . 31
5.2 Encoding . . . 32
5.2.1 Generator Matrix . . . 34
5.3 Classical Decoding Algorithm . . . 35
5.3.1 Geometric Connections . . . 35
5.3.2 Decoding Algorithm . . . 36
5.3.3 Geometric Decoding Example . . . 37

Chapter 6: Quantum Multivariate Polynomial Interpolation . . . 39
6.1 The Problem Definition . . . 39
6.2 Quantum Query Model . . . 40
6.2.1 Quantum Fourier Transform . . . 40
6.2.2 Phase Query . . . 40
6.3 More Preliminaries . . . 41
6.4 The Algorithm . . . 42

Chapter 7: Quantum Decoding Algorithm for Reed-Muller Codes . . . 43
7.1 Polynomial : Coefficient Vector :: Codeword : Message . . . 43
7.1.1 Adapting Polynomial Interpolation . . . 44
7.1.2 The Algorithm . . . 45
7.2 Two Types of Noisy Oracles . . . 46
7.2.1 Connection to Message Transmission . . . 46

Conclusion . . . 47
8.1 Further Work . . . 47

Appendix A: Finite Geometries . . . 49

Bibliography . . . 51

Index . . . 53

Abstract

This thesis is an analysis of the query complexity of quantum decoding algorithms for geometric error-correcting codes. Many quantum decoding algorithms for error-correcting codes have already been devised, and we study the robustness of such algorithms to noisy or unreliable oracles, given a fixed query complexity. In chapter 4, we define one type of noisy quantum oracle and proceed with an original analysis of the quantum decoding algorithm for the binary simplex code. The remainder of the thesis is dedicated to exploring and reinterpreting a class of generalized simplex codes, called Reed-Muller codes, in two ways: as finite geometries and as polynomials. Chapter 7, the main result, presents the transformation of Reed-Muller decoding into multivariate polynomial interpolation and the successful simplification of a generalized quantum multivariate polynomial interpolation algorithm for decoding Reed-Muller codes.

Dedication

Till Carina,
min allra käraste syster.

Chapter 1

Quantum Computing

In the following chapter, we will delve into the basics of quantum computing. Before proceeding, it is important to realize that quantum computing describes a fundamentally different *model of computation* than the computer systems currently on the market. Much like the first programmable computer was realized by Alan Turing far before anyone built his remarkable invention, we too study quantum computing at a time when only the most rudimentary quantum computers are available in practice.

Due to the complex nature of quantum mechanics, there are a plethora of misconceptions about quantum computing. One of the most widespread is that quantum computers gain computational speedups by *trying all possible solutions at once*. This is simply not true; the state of the quantum bits may be unknown, but they are still in only one state. We will go into this more later. Perhaps my favorite that I have heard upon starting my thesis is that *quantum computers are like classical computers but in trinary*. False, we cannot efficiently simulate a quantum computer on a classical one. Lastly, probably the most optimistic is that *quantum computers are faster at all tasks compared to classical computers*. Quantum computers are faster than classical computers at some, *but not all*, computations.

I outline these common misconceptions not to criticize them, but rather to encourage the reader to exert patience and care with the following section, as quantum computing is so fundamentally different from classical computing.

The quantum mechanical properties upon which quantum computers are based have been studied extensively by physicists; we will avoid discussing such details and instead take these properties for granted in order to focus on the information-theoretic behavior of this new model of computation.

1.1 Information Representation

Classical computers, i.e. the computers that we all know and love, run on *bits* or 1's and 0's. This is the fundamental unit of information in classical computers. In quantum computers, information is built upon an analagous concept, the *quantum bit* or *qubit*. We will discuss the defining properties of qubits, some of which may seem very different from those of bits.

In classical computing, each bit can be in one of two states–1 or 0. We denote *quantum states* by $|0\rangle$ and $|1\rangle$, pronounced "*ket zero*" and "*ket one*," respectively. This follows traditional *Dirac notation*. Unlike classical bits, qubits can be in a *superposition* between these two states. That is, a qubit can lie in one of the infinite states "between" $|0\rangle$ and $|1\rangle$. We describe a quantum state as follows: $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, where α and β are complex numbers and $|\alpha|^2 + |\beta|^2 = 1$. This describes a probability distribution over the quantum states $|0\rangle$ and $|1\rangle$ with *amplitudes* α and β that each take values from $\mathbb{C}$, the complex field. In this way, we can think of $|\psi\rangle$ as a vector in $\mathbb{C}^2$. The states $|0\rangle$ and $|1\rangle$ form an orthonomal basis in this complex vector space and are called the *computational basis states*.

Definition 1.1. (Qubit) A qubit is a unit vector of $\mathbb{C}^2$, denoted

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$$

where $\alpha, \beta \in \mathbb{C}$ and $|\alpha|^2 + |\beta|^2 = 1$. Equivalently, $|\psi\rangle$ can also be represented as

$$|\psi\rangle = \begin{bmatrix} \alpha \\ \beta \end{bmatrix}.$$

So how can we know which state a bit or qubit is in? In classical computing, we can simply read the bit to determine whether it's a 0 or a 1. Qubits are a little trickier. We cannot read $|\psi\rangle$ to determine its exact amplitudes, α and β. As soon as we measure $|\psi\rangle$, the superposition will collapse to either $|0\rangle$ with probability $|\alpha|^2$ or $|1\rangle$ with probability $|\beta|^2$. As soon as $|\psi\rangle$ has been measured to reveal some quantum state, $|\psi\rangle$ will permanently collapse to that state, i.e. $|\psi\rangle = 1|0\rangle + 0|1\rangle$ or $|\psi\rangle = 0|0\rangle + 1|1\rangle$. Every time we measure it thereafter, we will observe the same state that we measured the first time.

Remark 1.2. Suppose you are given a weighted coin and you are asked to determine, through measurement, the weights on heads and tails. Obviously, with enough flips, the law of large numbers states that your measurements would approach the true weights. Now, suppose you are only allowed to flip the coin once. The task of determining the weights becomes impossible. This is why we cannot measure the amplitudes of qubits only through measurement.

You may be wondering why quantum phenomenon are useful in the context of computation. It seems impossible to build a model of computation on a fundamental unit of information that is unknowable, immeasurable. The beauty of quantum computation lies in the *manipulation* of these immeasurable qubits such that by the time we measure them at the end, the result will inform us of the state they started in. Simply measuring the qubits before doing any transformations is fundamentally the same as running a random number generator on a classical computer and trying a random possible solution. Obviously, this is not very useful, so we must *transform* the qubits to say anything intelligent about the end result that we measure.

1.2 Quantum Logic Gates

As we discussed earlier, $|0\rangle = 1|0\rangle + 0|1\rangle$ and $|1\rangle = 0|0\rangle + 1|1\rangle$. We sometimes represent these quantum states as column vectors of the amplitudes:

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \text{ and } |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Thus, we can represent a single-qubit transformation by a 2-by-2 matrix where the first column is the image of $|0\rangle$ and the second column is the image of $|1\rangle$ under the transformation. For example, the quantum NOT gate can be realized as a matrix:

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

This takes a state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ and transforms it into $X|\psi\rangle = \beta|0\rangle + \alpha|1\rangle$. Equivalently, by matrix multiplication we see that

$$X|\psi\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \beta \\ \alpha \end{bmatrix}.$$

So, any transformation on a single qubit can be represented by a 2-by-2 matrix. What about the transformation that, when given $|0\rangle$ or $|1\rangle$, outputs an *equal superposition* of $|0\rangle$ and $|1\rangle$? This would essentially give us a fair coin. We might represent that as follows:

$$\widehat{H} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

There are a few problems with this transformation. First, applying this transformation, we get $\widehat{H}|0\rangle = 1|0\rangle + 1|1\rangle$, which means that $|\alpha|^2 + |\beta|^2 = 1^2 + 1^2 \neq 1$. Since

we view a quantum state as a probability distribution, the squares of the amplitudes must sum to 1. So, we must *normalize* the matrix, to get

$$\widehat{H} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

This transformation still isn't *unitary*, meaning it doesn't preserve the magnitude of the input vector. For example, $\widehat{H}|-\rangle = 0|0\rangle + 0|1\rangle$. This is not a valid state, because the squares of the amplitudes do not add to 1. So, we must fix this transformation to the following:

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

The quantum gate above (H) is called the *Hadamard transform* or *Hadamard gate*. We will precisely define this quantum gate later in this chapter, as it is absolutely critical for work in quantum algorithms. H acts on the basis vectors as follows: $H|0\rangle = (|0\rangle + |1\rangle)/\sqrt{2}$ and $H|1\rangle = (|0\rangle - |1\rangle)/\sqrt{2}$.

Both of these states are in an equal superposition between $|0\rangle$ and $|1\rangle$, but they are distinct states. These states come up rather often, so they have been given the special shorthand notations $|+\rangle$ and $|-\rangle$, respectively.

Definition 1.3 (Plus and Minus States)**.** The following states (pronounced "plus" and "minus") will be used ubiquitously throughout this thesis in their shorthand form:

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

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

The requirements that quantum transformations be reversible and normalized are encapsulated by the property that the matrix representation for any transformation must be *unitary*. That is, $U^\dagger U = I$ where $U^\dagger$ is the transpose of the complex conjugate of U and I is the 2-by-2 identity matrix. Within those requirements, we can construct any transformation we like. What about transforming multiple qubits?

1.3 Multiple Qubits

As we discussed, single qubits live in $\mathbb{C}^2$. In order to express a superposition over multiple qubits, say n qubits, we need to take some *tensor products*. An n-qubit state

can be represented as vector in $(\mathbb{C}^2)^{\otimes n} \cong \mathbb{C}^{2^n}$. For example, in a two qubit system, we have the following basis states:

$$|0\rangle \otimes |0\rangle,\ |0\rangle \otimes |1\rangle,\ |1\rangle \otimes |0\rangle,\ \text{and } |1\rangle \otimes |1\rangle.$$

All two-qubit states can be thought of as a vector in $\mathbb{C}^2 \otimes \mathbb{C}^2 = \mathbb{C}^4$. With that in mind, the four basis states above can also be written as[1]:

$$|00\rangle,\ |01\rangle,\ |10\rangle,\ \text{and } |11\rangle.$$

Thus, any two-qubit state can be represented as a linear combination of these basis states, namely $|\psi\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle$. In general, an n-qubit state can be represented as a linear combination of basis states,

$$|\psi\rangle = \sum_{i \in \mathbb{Z}_2^n} \alpha_i |i\rangle, \quad \alpha_i \in \mathbb{C}$$

where

$$\sum_{i \in \mathbb{Z}_2^n} |\alpha_i|^2 = 1.$$

The probability of observing a state $|i\rangle$ upon measuring $|\psi\rangle$ is $|\alpha_i|^2$.

We may measure qubits individually as well. For example, measuring just the first qubit gives us 0 with probability

$$\sum_{i \in \mathbb{Z}_2^{n-1}} |\alpha_{0i}|^2,$$

leaving the post-measurement state

$$|\psi'\rangle = \frac{\sum_{i \in \mathbb{Z}_2^{n-1}} \alpha_{0i}|\alpha_{0i}\rangle}{\sqrt{\sum_{i \in \mathbb{Z}_2^{n-1}} |\alpha_{0i}|^2}}. \tag{1.1}$$

1.4 Entangled States

Something that follows from the post-measurement equation (1.1) but which is not self-evident is the concept of *entangled states*. Consider the following state:

$$|\beta_{00}\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}}$$

[1] In this thesis, the following notations are equivalent: $|xy\rangle, |x, y\rangle$, and $|x\rangle \otimes |y\rangle$.

Suppose we measured the first qubit. We will observe $|0\rangle$ with probability 1/2, and the resulting post-measurement state is: $|\beta'_{00}\rangle = |00\rangle$ (by 1.1). How can this be? We only measured the first qubit, and yet, we now have information about both qubits. This is precisely because the state is entangled. Another example of an entangled state is:

$$|\beta_{01}\rangle = \frac{|01\rangle + |10\rangle}{\sqrt{2}}$$

These two states, $|\beta_{00}\rangle$ and $|\beta_{01}\rangle$, are the first two *Bell states* or *EPR pairs*. Quantum entanglement is a powerful computational tool and will be used in many quantum algorithms in the rest of the text.

Now that we have the means to represent an n-qubit state, the state transformations can be represented by 2^n-by-2^n matrices. One particularly useful transformation is the n-qubit Hadamard transform.

1.5 Hadamard Transform

As promised, a more precise definition of the Hadamard transform follows. Note that $H^{\otimes n} = H \otimes H \otimes \ldots \otimes H, n$ tensor products with itself.

Definition 1.4 (Hadamard Transform)**.** The n-qubit Hadamard transform, $H^{\otimes n}$ acts on a basis state $|k\rangle$ as follows:

$$H^{\otimes n}|k\rangle = \frac{1}{\sqrt{2^n}} \sum_{j \in \{0,1\}^n} (-1)^{k \cdot j} |j\rangle.$$

Upon first glance, this formula is quite uninviting, so I will provide a quick guide for some common states.

$$H^{\otimes n}|0\rangle^{\otimes n} = |+\rangle^{\otimes n}$$
$$H^{\otimes n}|1\rangle^{\otimes n} = |-\rangle^{\otimes n}$$

Since $H = H^{-1}$, we also have that

$$H^{\otimes n}|+\rangle^{\otimes n} = |0\rangle^{\otimes n}$$
$$H^{\otimes n}|-\rangle^{\otimes n} = |1\rangle^{\otimes n}.$$

This concludes the section on the Hadamard transform, though the careful (or confused) reader will find that they will have to refer back to this section in the coming chapters as the Hadamard transform is central to this thesis.

1.6 Measuring in Other Bases

Thus far, all of the measurements we have performed have been done in the *computational basis*. Given a state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, upon measurement in the computational basis $\{|0\rangle, |1\rangle\}$ this state will collapse to $|0\rangle$ with probability $|\alpha|^2$ and $|1\rangle$ with probability $|\beta|^2$. But, this choice of basis is arbitrary. We could instead measure $|\psi\rangle$ in the Hadamard basis, $|+\rangle$ and $|-\rangle$. Then, we can re-express the state $|\psi\rangle$ as follows

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle = \alpha\frac{|+\rangle + |-\rangle}{\sqrt{2}} + \beta\frac{|+\rangle - |-\rangle}{\sqrt{2}} = \frac{\alpha+\beta}{\sqrt{2}}|+\rangle + \frac{\alpha-\beta}{\sqrt{2}}|-\rangle.$$

So, measuring $|\psi\rangle$ in the Hadamard basis yields $|+\rangle$ with probability $|\alpha+\beta|^2/2$ and $|-\rangle$ with probability $|\alpha-\beta|^2/2$.

More generally, given an orthonormal basis $\{|a\rangle, |b\rangle\}$, we can represent any state $|\psi\rangle$ as a unique linear combination of this basis, $|\psi\rangle = \alpha|a\rangle + \beta|b\rangle$.Then, the probabilities of measuring $|a\rangle$ and $|b\rangle$ are $|\alpha|^2$ and $|\beta|^2$, respectively.

1.7 Phase

Up to this point, we have used the term *phase* slightly ambiguously. The exact meaning of the word differs depending on context, so we will flesh these out. In fact, some of the original results in this thesis puzzled both student and adviser until we revisited the exact distinction between *global phase* and *relative phase*.

Consider the states $|\psi\rangle$ and $e^{i\theta}|\psi\rangle$, where $|\psi\rangle$ is a state vector and θ is a real number. These two phases are equivalent up to their *global phase shift*. This just means that the probability of observing some basis state $|k\rangle$ under measurement in any basis is the same for both states. Observationally, they are identical states, so we can ignore global phase shifts.

The other notion of phase that will come up is *relative phase*. Consider the states $|+\rangle = (|0\rangle + |1\rangle)/\sqrt{2}$ and $|-\rangle = (|0\rangle - |1\rangle)/\sqrt{2}$. These two states have amplitudes on basis state $|1\rangle$ equal to $1/\sqrt{2}$ and $-1/\sqrt{2}$, respectively. So, although their magnitudes are equal, their signs are not. Now, observing both of these in the computational basis yields $|0\rangle$ with probability $1/2$ and $|1\rangle$ with probability $1/2$ for both states. In the computational basis they are indistinguishable; however, measuring in the *Hadamard basis*, with basis states $|+\rangle$ and $|-\rangle$, they are distinguishable with probability 1. Despite having the same relative phase, the states are in fact computationally distinct.

1.8 Quantum Oracles

Within any computational model, some computations are "expensive" for any number of reasons: they require large amounts of resources such as time, space, or circuitry. For this reason, one may wish to outsource such computations to third parties, which we will call *oracles*. As the name would suggest, oracles may be queried on inputs, and magically in one time step will output the result of the computation on the input, for a particular function. More concretely, an oracle $\mathcal{O}_f$ outputs $f(x)$ when queried on the input x.

Classical oracles are used throughout computer science theory to abstract computations and reason about algorithms and protocols. Quantum oracles differ from classical oracles in significant ways.

1.8.1 Quantum Queries

First, consider the action of the query. Suppose the function operates on the following spaces $f : \mathbb{Z}_2^n \to \mathbb{Z}_2^m$, and oracle $\mathcal{O}_f$ computes f. Clearly, if $n \neq m$, this function could not possibly be bijective. In order to ensure the reversibility of queries to $\mathcal{O}_f$, we must maintain two registers: a query register, and a response register. The query register contains the input x on which we wish to query the oracle and remains unchanged by the query. The response register is initialized to some known basis state $|r\rangle \in \mathbb{Z}_2^m$. After the quantum query takes place, the oracle records the response by taking the bitwise exclusive OR, or addition modulo two of $f(x)$ and r. Thus, the response register has the post-query state $|r \oplus f(x)\rangle$.

Definition 1.5 (Quantum Oracle Query)**.** Let $f : \mathbb{Z}_2^n \to \mathbb{Z}_2^m$ be a function. Then, the quantum oracle $\mathcal{O}_f : \mathbb{Z}_2^n \times \mathbb{Z}_2^m \to \mathbb{Z}_2^n \times \mathbb{Z}_2^m$ that computes f has the following action on the query and response registers

$$\mathcal{O}_f : |x, r\rangle \to |x, r \oplus f(x)\rangle.$$

In addition to the query and response registers, quantum oracles have the special ability to take as input a *superposition of queries* and output a *superposition of responses*. More precisely, suppose we want to query the oracle on the following valid superposition of points $S \subseteq \mathbb{Z}_2^n$.

$$|x\rangle = \sum_{s \in S} \alpha_s |s\rangle, \quad \text{where } \sum_{s \in S} |\alpha_s|^2 = 1.$$

Then, we need to append an m-qubit response register to each query register $|s\rangle$ and initialize the response registers to some known value $|r_s\rangle$. The pre-query state

becomes

$$\sum_{\substack{s\in S\\ r_s\in\mathbb{Z}_2^m}} \alpha_s|s,r_s\rangle.$$

Then, the query $\mathcal{O}_f : \mathbb{Z}_2^n \times \mathbb{Z}_2^m \to \mathbb{Z}_2^n \times \mathbb{Z}_2^m$ produces the following unitary transformation

$$\sum_{\substack{s\in S\\ r_s\in\mathbb{Z}_2^m}} \alpha_s|s,r_s\rangle \xrightarrow{\mathcal{O}_f} \sum_{\substack{s\in S\\ r_s\in\mathbb{Z}_2^m}} \alpha_s|s,r_s\oplus f(s)\rangle.$$

It is important to note that this only constitutes *one* quantum query to the oracle.

1.9 Phase-kickback Trick

One common query method is using what is called the *phase kickback trick*. The basic idea is that we initialize the response register to $|-\rangle$ so that both the query and response registers stay the same after the query, and the value of $f(x)$ is encapsulated by the phase of the state. More concretely, we have:

$$\begin{aligned}
|x\rangle\otimes|-\rangle &= |x\rangle\otimes\frac{1}{\sqrt{2}}(|0\rangle-|1\rangle)\\
&= \frac{1}{\sqrt{2}}(|x,0\rangle-|x,1\rangle)\\
&\xrightarrow{f} \frac{1}{\sqrt{2}}(|x,f(x)\rangle-|x,1\oplus f(x)\rangle\\
&= |x\rangle\otimes\frac{1}{\sqrt{2}}\left(|f(x)\rangle-|\overline{f(x)}\rangle\right)\\
&= (-1)^{f(x)}|x\rangle\otimes|-\rangle.
\end{aligned}$$

Since the response register remains unchanged, we will generally omit this from future computations, though technically it must be present to preserve the reversibility of the query.

1.10 Bernstein-Vazirani Algorithm

Now that we have seen a few tricks of the trade, we will dive into a quantum algorithm. The Bernstein-Vazirani algorithm forms the foundation for many of the algorithms in the rest of this thesis, so a solid grasp of this section will yield high dividends.

The Bernstein-Vazirani algorithm solves the following problem: for $N = 2^n$, we are given a function $f : \mathbb{Z}_2^n \to \mathbb{Z}_2$ with the property that there exists some unknown $a \in \mathbb{Z}_2^n$ such that $f(i) = i \cdot a \pmod 2$. The goal is to find a.

First, an overview of the algorithm: we will start in the state $|0\rangle^{\otimes n}$, the n-qubit zero state, and apply a n-qubit Hadamard transform to get an equal superposition of the states, i.e. $|+\rangle^{\otimes n}$. Next, we perform a quantum query using the phase kickback trick to store $f(i)$ in the phase of state $|i\rangle$. Then, another Hadamard transform on all n qubits. With a simple measurement, we obtain a with probability 1. Now, for the details.

$$|0\rangle^{\otimes n} \xrightarrow{H^{\otimes n}} \frac{1}{\sqrt{2^n}} \sum_{i \in \mathbb{Z}_2^n} |i\rangle \tag{1.2}$$

$$\xrightarrow{\mathcal{O}_f} \frac{1}{\sqrt{2^n}} \sum_{i \in \mathbb{Z}_2^n} (-1)^{f(i)} |i\rangle \tag{1.3}$$

$$\xrightarrow{H^{\otimes n}} \frac{1}{2^n} \sum_{i \in \mathbb{Z}_2^n} (-1)^{f(i)} \sum_{j \in \mathbb{Z}_2^n} (-1)^{i \cdot j} |j\rangle \tag{1.4}$$

$$= |a\rangle \tag{1.5}$$

This computation looks a mess for the beginner, so let's try to make sense of it. The first three lines follow from the definitions of the functions (confused readers may need to review the defined action of H and $\mathcal{O}$ with phase-kickback). The last step, however, is less clear-cut. Note that $(-1)^{f(i)} = (-1)^{(i \cdot a) \mod 2} = (-1)^{(i \cdot a)}$, given by $f(i) = i \cdot a \pmod 2$ in the problem statement. Thus, we may manipulate 1.4 as follows:

$$\frac{1}{2^n} \sum_{i \in \mathbb{Z}_2^n} (-1)^{f(i)} \sum_{j \in \mathbb{Z}_2^n} (-1)^{i \cdot j} |j\rangle = \frac{1}{2^n} \sum_{i \in \mathbb{Z}_2^n} (-1)^{i \cdot a} \sum_{j \in \mathbb{Z}_2^n} (-1)^{i \cdot j} |j\rangle \tag{1.6}$$

$$= \frac{1}{2^n} \sum_{j \in \mathbb{Z}_2^n} \sum_{i \in \mathbb{Z}_2^n} (-1)^{i \cdot (a+j)} |j\rangle \tag{1.7}$$

Now, suppose $j = a$ (note that $(-1)^{2k} = 1 \; \forall k \in \mathbb{Z}$). Then, from 1.7 we have

$$\begin{aligned} \frac{1}{2^n} \sum_{i \in \mathbb{Z}_2^n} (-1)^{i \cdot (2a)} |a\rangle &= \frac{1}{2^n} \sum_{i \in \mathbb{Z}_2^n} (-1)^{2(i \cdot a)} |a\rangle \\ &= \frac{1}{2^n} \sum_{i \in \mathbb{Z}_2^n} 1 |a\rangle \\ &= |a\rangle. \end{aligned}$$

So, 1.7 is simply the state $|a\rangle$; thus, measurement in the computational basis will yield a with probability 1. This algorithm is the bread and butter of this thesis, so to speak, and is critical to the decoding algorithms we will study in a few chapters.

1.11 Advanced Topics

In this section, we discuss some advanced topics that are not entirely necessary for understanding this thesis, but which are nonetheless vital to understanding quantum computation as a whole.

1.11.1 No-Cloning Theorem

Suppose my friend has a secret state $|\psi\rangle$ that she wants me to have a copy of. Classically, we could easily build a circuit to copy her state $x = 0$ or $x = 1$ without measuring it. We would simply initialize a temporary register to 0, and our circuit would write $x \oplus 0$, the XOR of x and 0, to the destination register.

The quantum case is a bit trickier. Suppose $|\psi\rangle$ is only known to our friend. We wish to copy this exact state into a target slot, which starts out in some standard known state $|s\rangle$. Thus, the initial state of our copying machine is $|\psi\rangle \otimes |s\rangle$. Now, we apply some unitary operation U to these registers to obtain

$$|\psi\rangle \otimes |s\rangle \xrightarrow{U} U(|\psi\rangle \otimes |s\rangle) = |\psi\rangle \otimes |\psi\rangle \tag{1.8}$$

Now, suppose this quantum copying circuit works for two arbitrary states $|\psi\rangle$ and $|\phi\rangle$. Then, we have

$$U(|\psi\rangle \otimes |s\rangle) = |\psi\rangle \otimes |\psi\rangle \tag{1.9}$$

and

$$U(|\phi\rangle \otimes |s\rangle) = |\phi\rangle \otimes |\phi\rangle \tag{1.10}$$

Now, taking the inner product of the left-hand sides of equations 1.9 and 1.10 gives

$$(\langle\psi| \otimes \langle s|)U^\dagger U(|\phi\rangle \otimes |s\rangle) = (\langle\psi|U^\dagger U|\phi\rangle)(\langle s|U^\dagger U|s\rangle) \tag{1.11}$$

$$= \langle\psi \mid \phi\rangle \tag{1.12}$$

However, taking the inner products of the right-hand sides of equations 1.9 and 1.10 gives

$$(\langle\psi| \otimes \langle\psi|)(|\phi\rangle \otimes |\phi\rangle) = (\langle\psi \mid \phi\rangle)^2 \tag{1.13}$$

Now, equations 1.12 and 1.13 give

$$\langle\psi \mid \phi\rangle = (\langle\psi \mid \phi\rangle)^2.$$

However, this equation is only true if $|\psi\rangle = |\phi\rangle$ or if $|\psi\rangle$ is orthogonal to $|\phi\rangle$. Thus, a general cloning device can only clone states that are orthogonal, which means that we cannot clone two arbitrary states.

This is an important fundamental difference between the classical and quantum models of computation. We take for granted in classical computing that we can copy any unknown state, and much of classical computation relies upon this fact. Another fact of classical computing which we take for granted is the existence of universal gates.

1.11.2 Universal Quantum Gate

In classical computation, there are three basic logic gates: NOT, AND, and OR. We can combine these gates to represent any possible logical expression. Furthermore, the NAND gate and the NOR gate, which we get from taking the negation (NOT) of the output of AND and OR respectively, are said to be *universal gates*. This means that we can construct the three basic logic gates just from NAND gates or just from NOR gates. So, NAND and NOR are universal in that any logical expression can be represented with just NAND or NOR circuits. This is very useful in practice because NAND gates are very cheap to construct, so using only NANDs can keep the cost of a computer chip down.

A natural question, then, is whether there exists a quantum analogue, a universal quantum gate. The short answer is that there is no single universal quantum gate, however the following three gates are enough to make an *arbitrarily good approximation* of any quantum gate [Nielsen and Chuang, 2002]: Hadamard, CNOT, and phase gate.

The phase gate is defined

$$\begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix} \tag{1.14}$$

And the CNOT, or controlled-NOT gate acts on two qubits by flipping the second qubit if and only if the first qubit is a 1.

$$\text{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{1.15}$$

As you can see, by multiplying the CNOT matrix by the column vector $(a \quad b \quad c \quad d)^\intercal$ this takes $a|00\rangle+b|01\rangle+c|10\rangle+d|11\rangle$ and transforms it into $a|00\rangle+b|01\rangle+c|11\rangle+d|10\rangle$, thereby swapping the amplitudes on the states $|10\rangle$ and $|11\rangle$, or equivalently "flipping" the second bit of the state if the first bit is a 1.

The *gate complexity* of a unitary transformation is the minimum number of gates needed to implement the circuit.

Chapter 2

Error-Correcting Codes

No real-life communication channel is perfect: internet packets occasionally get dropped, radio transmissions are drowned out by static interference, and data on storage media get corrupted. Distinguishing signal from noise is a problem as old as communication itself. One way of adapting to this problem is introducing redundancy into the transmission. If the information we want to transmit is encoded in more than one place, then there is a higher probability that at least one of the copies will make it through even if some of the copies are plagued with errors. This is precisely the approach that error-correcting codes take.

Error-correcting codes were developed to enable the reliable transmission of messages over noisy communication channels. There are many examples of their use today, and many error-correcting codes have been developed for specific noise models. In CDs[1], error-correcting codes prevent (small) scratches from leading to data loss. They are also used for transmitting photographs from rovers and telescopes throughout the solar system to earth.

Error-correcting codes are important for the practical implementation of quantum computers because current physical implementations of quantum computers are plagued by *quantum decoherence*. That is, the qubits are difficult to fully isolate from their environment and thus become entangled with their surroundings, leading to noisy computations. For now, we will forget about quantum computing and plunge into the code theory.

2.1 Preliminaries

There are many kinds of communication channels, but we will focus on *binary symmetric channels*.

[1] If they still exist...

Definition 2.1 (Binary Symmetric Channel)**.** A binary symmetric channel is a classical channel that can transmit a string of 0's and 1's. If a bit b is sent, b will be flipped to $\neg b$ with probability $p \in [0, 1]$, and b will be transmitted correctly with probability $1 - p$ where $p < \frac{1}{2}$.

This type of channel is *binary* because we are transmitting bits, and it is *symmetric* because there is an equal probability of a 0 flipping to a 1 and the reverse. Note that if our binary symmetric channel flips bits with probability $p > 1/2$, we could easily create a channel with $p < 1/2$ by negating every bit on the receiving or sending end. Additionally, note that if $p = 1/2$, it is information-theoretically impossible to recover the message, so we don't consider this case and frankly it is dubious to even call that a communication channel.

In the following sections we will investigate how to mitigate the information loss from random errors in binary symmetric channels where $p < 1/2$. In the following chapter, and throughout the thesis, we will use the terms "noise" and "errors" interchangeably.

2.2 Overview of Transmission

Thus far, we have only described the parts of the communication system in vague terms; now, we will assuage the reader's thirst for clarity and precision. We will, of course, delve further into each of these parts later.

There are five basic components in the communication system we consider. The *message source*, or *sender*, transfers a message $m = m_1 \dots m_k$ to the *encoder*. The *encoder* outputs the *codeword* $c = c_1 \dots c_n$ associated with message m and passes it along to the *binary symmetric channel*. The *channel* incorporates an *error vector* $e = e_1 \dots e_n$ into the codeword and outputs the received vector $r = e + c = r_1 \dots r_n$ where the addition is bitwise and modulo 2. Then, the *decoder* takes the received vector (also referred to as the "noisy codeword") r and outputs $\hat{m}$, an estimate of message m to the *receiver*. A schematic is provided below.

$$\text{Sender} \xrightarrow{\text{Message } m} \text{Encoder} \xrightarrow{\text{Codeword } c} \text{Channel} \xrightarrow{\text{Rcvd. Vect. } m+e} \text{Decoder} \xrightarrow{\text{Est. } \hat{m}} \text{Receiver}$$

2.3 Encoding

The job of the encoder is to introduce well-defined redundancy into the transmission. We will describe a general encoding scheme that applies to all linear codes, though the more advanced encoding algorithms we will explore later will deviate from this general form.

In this general code $\mathscr{C}$, the message will be encoded as a codeword that consists of a series of *check symbols* appended to the original message m. The check symbols encode redundancy into the codeword. More concretely, the bits of the codeword c associated with message m are defined

$$c_1 = m_1, \quad c_2 = m_2, \quad \ldots, \quad c_k = m_k$$

and the last $n - k$ bits of c are all check symbols $c_{k+1} \ldots c_n$. The check symbols are determined by the codeword, and they are chosen so that

$$P \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} = P\mathbf{c}^{tr} = 0$$

where P is the *parity check matrix* for the code $\mathscr{C}$.

Definition 2.2 (Parity Check Matrix)**.** The parity check matrix P for a linear code $\mathscr{C}$ is given by

$$P = [A \mid I_{n-k}]$$

where A is some fixed $(n-k) \times k$ binary matrix and I_{n-k} is the $(n-k) \times (n-k)$ identity matrix.

A code $\mathscr{C} \subset \mathbb{Z}_2^n$ is simply the set of all codewords c that satisfy the equation $P\mathbf{c}^{tr} = 0$. In other words, $\mathscr{C} = \{c \in \mathbb{Z}_2^n \mid P\mathbf{c}^{tr} = 0\}$.

Example 2.3. Suppose our parity check matrix is

$$P = \left[\begin{array}{ccc|cc} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \end{array}\right]$$

This defines a code $\mathscr{C}$ with $n = 5$ and $k = 3$. For this code,

$$A = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}$$

Moreover, the message $m = m_1 m_2 m_3$ is encoded into a codeword $c = c_1 c_2 c_3 c_4 c_5$ such that $c_1 = m_1$, $c_2 = m_2$, and $c_3 = m_3$ and then

$$\begin{aligned} c_4 &= -(c_2 + c_3) \equiv c_2 + c_3 \pmod 2 \\ c_5 &= -(c_1 + c_3) \equiv c_1 + c_3 \pmod 2 \end{aligned}$$

Given a parity check matrix and a message, the encoder needs to output the corresponding codeword. This is best done with the *generator matrix*.

Definition 2.4 (Generator Matrix)**.** Given a parity check matrix $P = [A \mid I_{n-k}]$ for a code $\mathscr{C}$, the corresponding generator matrix is $G = [I_k \mid A^{tr}]$. To generate the codeword c for a message m, simply multiply the matrices $c = mG$.

Remark 2.5. Note that a given code $\mathscr{C}$ may have more than one matrix that generates it.

Example 2.6. Continuing with the code defined in example 2.3, we have that the corresponding generator matrix for this code is

$$G = \left[\begin{array}{ccc|cc} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \end{array}\right]$$

2.4 Decoding

As was alluded to, the channel will (term-wise) add some unknown error vector $e = e_1 \dots e_n$ to the codeword so that the decoder receives a "noisy codeword" or "received vector" $r = m + e \pmod 2$.

The decoder has a significantly harder job than the encoder. Namely, it must take this noisy codeword r and return the message m that produced the vector. Any given decoding algorithm will not always be successful; sometimes there will be so many errors in a given received codeword that it is impossible to parse which message was sent. That said, some decoding algorithms are more effective than others, and this distinction comes down to the properties of the code $\mathscr{C}$.

2.4.1 Properties of Vectors

One of the key defining properties of a vector is its Hamming weight.

Definition 2.7 (Hamming weight)**.** The Hamming weight of a vector $x = x_1 \dots x_n$ is the number of nonzero x_i, and is denoted by $wt(x)$.

Example 2.8. For example, the vector $x = 0110100111$ has Hamming weight 6.

Another important property in relating two vectors is their *Hamming distance*.

Definition 2.9 (Hamming distance)**.** The Hamming distance of two vectors $x = x_1 \dots x_n$ and $y = y_1 \dots y_n$ is defined $dist(x, y) = wt(x \oplus y)$, where the XOR is bitwise.

Note that one could write this definition in many ways (i.e. addition or subtraction mod 2), but the heuristic idea behind computing the Hamming distance between two vectors is to add up how many of their bits differ. This concept is a fundamental building block for the decoding algorithms we will encounter.

Example 2.10. dist(010101, 100111) = 3

2.4.2 Errors

Earlier we defined the error vector $e = e_1 \dots e_n$ where $e_i \in \mathbb{Z}_2$. The received vector is defined $r = m + e$, which implies that $e_i = 1$ if and only if the i-th bit of the codeword was corrupted in transmission and $e_i = 0$ otherwise. Since the binary symmetric channel corrupts a single bit with probability p, it follows that $P[e_i = 1] = p$. One may wonder: *what is the most probable error vector?* This would certainly help the decoder to determine which codeword was sent given the noisy codeword.

Since the corruption of bits by the communication channel are independent of one another, we can compute the probability that the error vector e is some particular vector x:

$$P[e = x] = \prod_{i=1}^{n} P[e_i = x_i] = p^{wt(x)}(1-p)^{n-wt(x)}. \tag{2.1}$$

This perhaps goes without saying, but the error vector is entirely independent of the message.

Example 2.11. Suppose $n = 5$. Then, $P[e = 10110] = p(1-p)p^2(1-p) = p^3(1-p)^2$.

So, now we want to maximize the equation

$$P[e = x] = \prod_{i=1}^{n} P[e_i = x_i] = p^{wt(x)}(1-p)^{n-wt(x)}$$

to figure out the most likely error vector x and thus the best decoding algorithm. We have that $p < \frac{1}{2}$, so it follows that the maximum probability is achieved when $n - wt(x)$ is as large as possible. This occurs when $x = \mathbf{0}$, the zero vector. In other words, *the most likely error vector is one that indicates that none of the bits were corrupted.*

In terms of decoding, this fact implies that the optimal decoding algorithm assumes that the weight of e is as small as possible. If the received vector r is not a codeword, then the decoder should output the nearest codeword c with respect to the Hamming distance. This decoding method is called *nearest neighbor decoding* and it is the strategy we will employ.

2.4.3 Properties of Codes

Now that we have defined the nearest neighbor decoding strategy, one may wonder how many errors a given code can correct. Before proceeding, note the distinction between *error detection* whereby the code informs the decoder that x number of errors occurred and *error correction* whereby the decoder is able to correct the x errors. Analyzing the error-correcting and error-detecting properties of linear codes will require some additional metrics on codes.

One of the traits of a code that is most pertinent to its error-correcting properties is called its *minimum distance*. This captures the worst-case correction performance of a code on a nearest neighbor decoding algorithm.

Definition 2.12 (Minimum Distance)**.** The minimum distance d of a code $\mathscr{C}$ is

$$d(\mathscr{C}) = \min_{x,y \in \mathscr{C}} \{dist(x, y)\}.$$

The minimum distance of a code $\mathscr{C}$ encapsulates how closely packed the codewords are in n-dimensional space. This determines how many errors we can correct with nearest-neighbor decoding.

Theorem 2.13. A linear code $\mathscr{C}$ with minimum distance d can correct $\lfloor \frac{1}{2}(d-1) \rfloor$ errors.

Proof. Suppose our code $\mathscr{C}$ has minimum distance d. Now, imagine each codeword has an n-dimensional sphere[2] of radius t around it. Now, if t is small enough, none of the spheres will overlap. This means that a received vector r with at most t errors will be decoded correctly using nearest neighbor decoding. Now, the question is: what is the largest that the radius t can be and still maintain the non-overlapping n-spheres? This is precisely how error-correcting codes can be translated to sphere packing problems.

Anyway, if $d = 2t + 1$, then spheres with radius t will not overlap. Thus, we have that $t = \frac{1}{2}(d-1)$ is the largest that t may be. Now, we can correct up to t errors since the n-spheres do not overlap, but the number of errors in a codeword are discrete integer values. Thus, we can only guarantee that $\lfloor t \rfloor$, or $\lfloor \frac{1}{2}(d-1) \rfloor$ errors can be corrected.

□

2.4.4 Decoding Errors

One may wonder what happens if there are more than $\lfloor \frac{1}{2}(d-1) \rfloor$ errors occur during transmission. In this case, the received vector $r = e + c$ may be closer to some

[2]Or, for the topologists among you, let's call it the closed n-ball $\overline{B(c,t)}$ where $c \in \mathbb{Z}_2^n$ is a codeword.

other codeword c', so nearest neighbor decoding will decide that c' was the original codeword rather than c, the actual codeword that was sent. This is called a *decoding error* because the decoder erroneously returned c' instead of c. This is not ideal, and code theorists study the probability of this occurrence, denoted P_{err}.

[****If I want to talk about this then I have to introduce cosets and syndromes and I don't feel like doing that so let's just skip it]

Chapter 3

Learning Theory

Given oracle ("black box") access to some function f, how many queries would it take to determine some property of f? This is precisely the question that learning theory is concerned with. This field is motivated in part by the idea that the oracle computes some "expensive" function, i.e. one that is time or space intensive. Minimizing the number of queries that an algorithm has to make to an oracle keeps the computational cost expended on oracle queries low.

3.1 Basics of Concept Learning

Consider the oracle function f. This is a *Boolean function*, as it takes inputs in $\{0,1\}^n$ for some fixed n and outputs 0 or 1. In the field of learning theory, f is called a *concept*.[1]

Definition 3.1 (Concept)**.** A concept f over $\{0,1\}^n$ is a Boolean function $f : \{0,1\}^n \to \{0,1\}$. Equivalently, a concept can be viewed as a subset of $\{0,1\}^n$, namely $f = \{x \in \{0,1\}^n \mid f(x) = 1\}$.

Computer scientists do not care how many queries it takes to learn the properties of some particular concept, but they are more interested in knowing how the number of queries *scales asymptotically* with the size of the input. To take care of this, they came up with *concept classes*. Traditionally, concept classes are denoted by $\mathcal{C}$, but we will instead call them $\mathcal{F}$ to avoid confusion with codes $\mathscr{C}$.

Definition 3.2 (Concept Class)**.** A concept class $\mathcal{F} = \cup_{n \geq 1} f_n$ is a collection of concepts where $f_n = \{f \in \mathcal{F} \mid f \text{ is a concept over } \{0,1\}^n\}$.

[1] CITE Quantum vs classical learnability by Servedio!!!

Example 3.3. Suppose the domain we are working with is sets of 2^n people instead of $\{0,1\}^n$. Then, we can define a concept like $f = \{x \in 2^n\text{-set of people} \mid \text{Marika has met } x\}$. Then, $f(\text{“Jamie Pommersheim”}) = 1$, but $f(\text{“Alan Turing”}) = 0$, sadly. Now, the concept class is just the union of these concepts over all possible domain sizes (ignoring the fact that the number of people that have existed is finite). This example is perhaps not mathematically rigid, but it should give the reader an intuition for what concept classes are.

The example we outlined refers to a specific kind of function called a *membership oracle*.

Definition 3.4 (Membership Oracle)**.** A membership oracle MQ_f is an oracle that returns 1 on input x if and only if $x \in f$ or equivalently if $f(x) = 1$. This is called a *membership query*.

Given some membership oracle MQ_f, the goal of the learning algorithm is to construct a *hypothesis* $h : \{0,1\}^n \to \{0,1\}$ that is equivalent to f, i.e. $\forall x \in \{0,1\}^n$, $f(x) = h(x)$. While their outputs are equal, f and h may compute their outputs quite differently, but this is irrelevant.

Chapter 4

The Binary Simplex Code

One particularly interesting linear code is the simplex code. This is a special case of the Reed-Muller codes we will discuss later. The encoding and decoding algorithms are quite simple, so this is our starting point for further analyzing some of the error-correcting properties of the quantum decoding algorithm.

4.1 Encoding

To encode a message m with n bits, simply append the inner product of each binary number of length n (in order) with the message. The 2^n bits of codeword c for message m are:

$$c_i = i \cdot m \pmod 2 \quad i \in \mathbb{Z}_2^n.$$

Example 4.1. Suppose $n = 3$. Then, the encoding of the message $m = 010$ is:

$$\begin{aligned}
c_1 &= (0,0,0) \cdot (0,1,0) = 0 \\
c_2 &= (0,0,1) \cdot (0,1,0) = 0 \\
c_3 &= (0,1,0) \cdot (0,1,0) = 1 \\
c_4 &= (0,1,1) \cdot (0,1,0) = 1 \\
c_5 &= (1,0,0) \cdot (0,1,0) = 0 \\
c_6 &= (1,0,1) \cdot (0,1,0) = 0 \\
c_7 &= (1,1,0) \cdot (0,1,0) = 1 \\
c_8 &= (1,1,1) \cdot (0,1,0) = 1
\end{aligned}$$

Thus, $c = 00110011$ for that message.

Now, the computer scientists reading this will notice that the length of the encoding of a message is exponential in the length of the message, i.e. a message of length n has a codeword of length 2^n. This is far from an ideal representation of information. Indeed, the codeword for a 256-bit message would be 2^{253} bytes long, which is so large that our units of storage do not go that high. For some perspective, the largest unit of storage I could find is the yottabyte, or 2^{80} bytes, or one trillion terabytes. The length of the codeword, 2^{253} bytes, is far more information than could possibly be stored in the observable universe even if each bit were represented by a single atom. Clearly, this scheme is not ideal for long messages, but in the spirit of conducting unfettered computer science theory research, we will ignore this fact.

4.1.1 Generating Matrix

Another way of encoding messages with the binary simplex code is to use a generating matrix. Recall the Hadamard matrix.

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

Now, we will define a similar matrix transformation that is, in fact, the generating matrix for the binary simplex code.

$$S = \frac{1}{\sqrt{2}} \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \tag{4.1}$$

Note that the terms of S are equivalent to the terms of H with the following bijection: $h_{ij} = (-1)^{s_{ij}}$ (where h_{ij} is the term of H in row i column j). This property is essential to the quantum decoding algorithm.

Now, to create the generating matrix for a simplex code of length $n = 2^N$ for some $N \in \mathbb{Z}_{>0}$, simply take N tensor products of matrix 4.1, or $S^{\otimes N}$.

Remark 4.2. For the geometers among you, you will be pleased to know that the codewords of the binary simplex code do, in fact, lie on the vertices of an n-dimensional regular simplex as the name would suggest.

4.2 Quantum Decoding Algorithm

The quantum decoding algorithm for the simplex code is very pleasing. The Bernstein-Vazirani algorithm is the essence of the algorithm. In this version of the algorithm, we will assume that no errors occurred during the transmission of the codeword; later, we will analyze the performance of the algorithm with errors.

4.2.1 Reliable Quantum Oracle

The quantum oracle we will use is specific to each message m. That is, the oracle $\mathcal{O}_m$ encodes the message m into the corresponding codeword c. This oracle is "reliable" in the sense that it always gives c with no errors, i.e the received codeword is the actual codeword.

Precisely, $\mathcal{O}_m$ is a bijection on $(\mathbb{C}^2)^n \otimes \mathbb{C}^2$ and is defined as follows

$$\mathcal{O}_m : |i, r\rangle \longmapsto |i, r \oplus (i \cdot m)\rangle.$$

This returns precisely the i-th bit of the codeword c that corresponds to message m. Now, we may put $|-\rangle$ in the response register and use the phase-kickback trick to store c_i in the phase of the state:

$$\mathcal{O}_m : |i, -\rangle \longmapsto (-1)^{i \cdot m} |i, -\rangle.$$

In the algorithm, we will omit the response register since it remains $|-\rangle$ throughout the computation.

4.2.2 Algorithm

Now that our query model is defined, we give an overview of the algorithm. The n-qubit query register will be initialized with $|0\rangle^{\otimes n}$ and an n-qubit Hadamard transform will put the query register into an equal superposition. Simultaneously, the n-qubit response register will be set to $|-\rangle^{\otimes n}$ so we can use the phase-kickback trick. The algorithm does one quantum query to the oracle $\mathcal{O}_m$ to read in the corresponding codeword's bits into the phase of each state in the query register. Then, an n-qubit Hadamard is applied to the query register. Upon measurement, we obtain the original message with probability 1.

Now, for the algorithm. Let $|\psi_m\rangle$ denote the query register.

$$|\psi_m\rangle = \frac{1}{\sqrt{2^n}} \sum_{i \in \mathbb{Z}_2^n} |i\rangle \tag{4.2}$$

$$\xrightarrow{\mathcal{O}_m} \frac{1}{\sqrt{2^n}} \sum_{i \in \mathbb{Z}_2^n} (-1)^{i \cdot m} |i\rangle \tag{4.3}$$

$$\xrightarrow{H^{\otimes n}} \frac{1}{2^n} \sum_{i \in \mathbb{Z}_2^n} (-1)^{i \cdot m} \sum_{j \in \mathbb{Z}_2^n} (-1)^{i \cdot j} |j\rangle \tag{4.4}$$

$$= \frac{1}{2^n} \sum_{j \in \mathbb{Z}_2^n} \left(\sum_{i \in \mathbb{Z}_2^n} (-1)^{i \cdot (m+j)} \right) |j\rangle \tag{4.5}$$

$$= |m\rangle \tag{4.6}$$

Most of these transformations follow directly from definitions. The equality from line 4.5 to 4.6 is less apparent, but this equality follows the same logic as the analogous part of the Bernstein-Vazirani algorithm. Note that on line 4.5 if $j = m$, the inner sum becomes

$$\begin{aligned} \sum_{i \in \mathbb{Z}_2^n} (-1)^{i \cdot (2m)} &= \sum_{i \in \mathbb{Z}_2^n} 1 \\ &= 2^n. \end{aligned}$$

Thus, the full state on line 4.5 is

$$\begin{aligned} \frac{1}{2^n} \sum_{j \in \mathbb{Z}_2^n} \left(\sum_{i \in \mathbb{Z}_2^n} (-1)^{i \cdot (m+j)} \right) |j\rangle &= \frac{1}{2^n} \left(2^n |m\rangle + \sum_{\substack{j \in \mathbb{Z}_2^n \\ j \neq m}} (-1)^{i \cdot (m+j)} |j\rangle \right) \\ &= \frac{1}{2^n} \left(2^n |m\rangle + \sum_{\substack{j \in \mathbb{Z}_2^n \\ j \neq m}} 0 |j\rangle \right) \\ &= |m\rangle. \end{aligned}$$

So, measuring in the computational basis gives us c with probability 1. This algorithm yields the correct message m with probability 1 using only one quantum query.

4.3 Rosbustness to Errors

Now we will consider a different scenario. Suppose that the oracle $\mathcal{O}_m$ is a little faulty. The oracle occasionally makes mistakes in transmitting the bits of the codeword c that corresponds to m. Instead, it transmits a vector $r = c+e$ which may not even be a codeword in the simplex code. Since the simplex code is error-correcting, one would hope that the algorithm could still recover the correct m with a high probability. In this section, we formalize the algorithm's robustness to errors, that is, the probability of producing the correct output even with a noisy oracle.

Remark 4.3. Since this algorithm only requires one query to the oracle, we will not distinguish between a stateless and stateful noisy quantum oracle; however, that is to come.

4.3.1 Noisy Quantum Oracle

The reliable oracle is defined above. Now, we will define the unreliable, or noisy, quantum oracle. Again, it is a bijection on $(\mathbb{C}^2)^n \otimes \mathbb{C}^2$. Now, define a set of indices $E \subseteq [N]$, where $N = 2^n$, on which the oracle answers incorrectly. Then, the action of the noisy oracle $\widehat{\mathcal{O}}_m$ is defined as follows:

$$\widehat{\mathcal{O}}_m : |i, -\rangle \longmapsto (-1)^{m \cdot i} - 2\delta_E (-1)^{m \cdot i}$$

where δ_E is simply an indicator function on the set E. In other words, if $i \notin E$ then the oracle $\widehat{\mathcal{O}}_m$ returns the i-th bit of the codeword associated with m properly in the form $(-1)^{m \cdot i}$. On the other hand, if $i \in E$, the oracle returns $-(-1)^{m \cdot i}$, which is the i-th bit of the codeword flipped.

4.3.2 Algorithm with Noisy Oracle

Theorem 4.4. (Swanberg and Pommersheim) The Bernstein-Vazirani quantum decoding algorithm for the binary simplex code $\mathcal{S}_n$ has probability of success

$$P_{succ} = \left(1 - \frac{k}{2^{n-1}}\right)^2$$

where $k = |E|$, the number of errors introduced by the noisy quantum oracle and n is the length of the message.

Proof. We will begin by performing the algorithm with our noisy oracle $\widehat{\mathcal{O}}_m$.

$$\begin{aligned}
|\psi_m\rangle &= \frac{1}{\sqrt{2^n}} \sum_{i \in \mathbb{Z}_2^n} |i\rangle \\
&\xrightarrow{\widehat{\mathcal{O}}_m} \frac{1}{\sqrt{2^n}} \sum_{i \in \mathbb{Z}_2^n} (-1)^{i \cdot m} |i\rangle - 2 \left(\sum_{p \in E} (-1)^{p \cdot m} |p\rangle \right) \\
&\xrightarrow{H^{\otimes n}} |m\rangle - \frac{1}{2^{n-1}} \left(\sum_{p \in E} (-1)^{p \cdot m} \sum_{j \in Z_2^n} (-1)^{j \cdot p} |j\rangle \right) \\
&= |m\rangle - \frac{1}{2^{n-1}} \left(\sum_{p \in E} \sum_{j \in Z_2^n} (-1)^{p \cdot (j+m)} |j\rangle \right)
\end{aligned}$$

Now, we want to know P_{succ}, which is the probability of measuring m in the computational basis. We have that for any quantum state, the probability of measuring a given result is the square of the coefficient on that state. So,

$$\begin{aligned}
P_{succ} &= \left((1 - \frac{1}{2^{n-1}} \left(\sum_{p \in E} (-1)^{2(p \cdot m)} \right) \right)^2 \\
&= \left(1 - \frac{1}{2^{n-1}} \left(\sum_{p \in E} 1 \right) \right)^2 \\
&= \left(1 - \frac{|E|}{2^{n-1}} \right)^2 \\
&= \left(1 - \frac{k}{2^{n-1}} \right)^2 .
\end{aligned}$$

□

Interestingly, this formula shows that the quantum binary simplex decoding algorithm has $P_{succ} = 1$ if $k = 2^n$. In other words, if the quantum oracle is so noisy that it flips every single bit of the codeword, the decoding algorithm is still able to compute the correct message. This puzzled both the student and adviser on this thesis until we realized that flipping all of the bits leads to a *global phase shift* in the quantum state post-query, which as was discussed in Chapter 1, is essentially an equivalent state. So, in this case the adage that *it's so wrong it's right* holds true.

Additionally some features of the formula in theorem 4.4,

Corollary 4.5. (Swanberg and Pommersheim) The probability of success for the quantum decoding algorithm for the binary simplex code $\mathcal{S}_n$:

1. Does not depend on the message m, and
2. Does not depend on E, only $|E|$.

Proof. Corollary 4.5 follows from the proof of Theorem 4.4. □

Chapter 5

Reed-Muller Codes

Reed-Muller codes are error-correcting codes with fascinating mathematical properties. They are used in deep space communications to reliably transmit messages across space and time. Reed-Muller codes may be one of the oldest and most well-studied codes.

While the binary simplex code (singular) only had one parameter, namely the block length, the Reed-Muller codes (plural) have two parameters. In fact, Reed-Muller codes are a generalization of the binary simplex code (also called the Walsh-Hadamard code) and are closely related to other important codes. They belong to the family of *geometrical codes*, as evidenced by their close relationship with projective and Euclidean geometries (for which there is an appendix), and fortunately they are easier to decode than other geometrical codes. While they are used in some instances, such as for deep space communications, they have a smaller minimum distance than BCH (Bose-Chaudhuri-Hocquenghem) codes of the same block length.

5.1 Boolean Functions

One interesting way to conceive of Reed-Muller (RM) codes is as polynomials of Boolean functions. Let the block length $N = 2^n$ for some $n \in \mathbb{Z}^+$. The input to a RM Boolean function will be n variables taking binary values $v_1, \ldots, v_n \in \mathbb{Z}_2$. More precisely,

Definition 5.1. (Boolean function) Any function $f : \mathbb{Z}_2^n \to \mathbb{Z}_2$ which takes n binary input values $v_1, \ldots, v_n$ and outputs 0 or 1 is a Boolean function.

A Boolean function f can be defined in terms of its truth table, that is, a simple enumeration of what f evaluates to on all 2^n inputs.

Example 5.2. Suppose $N = 2^3$. Then, we may represent a Boolean function $f : \mathbb{Z}_2^3 \to \mathbb{Z}_2$ with its truth table. For example, we could define the following function.

$v_1v_2v_3$	$f(v_1, v_2, v_3)$
000	0
001	0
010	0
011	1
100	1
101	0
110	0
111	0

Now, a more compact way of representing the same function above is by simply writing $f = 00011000$.

All of the normal Boolean operations apply to Boolean functions, with the following modifications: $f \oplus g = f + g, f \wedge g = fg$, and $\neg f = 1 + f$. In particular, Boolean functions can be written as a polynomial of some particular functions which we will denote $v_1, v_2, \ldots v_n$. This notational overlap is not accidental, for v_i is a function which simply evaluates to whatever the i-th bit of the input is. In general, we can write $v_i = (0^{2^{i-1}}1^{2^{i-1}})^{2^{n-i}}$ where we use the computer science notation for concatenation as multiplication.

Example 5.3. For $N = 2^3$ again, then the following are the special functions v_1, v_2, v_3, which we will care about later, written in the notation defined above :

$$v_1 = 01010101$$
$$v_2 = 00110011$$
$$v_3 = 00001111$$

We can write any Boolean function $f : \mathbb{Z}_2^n \to \mathbb{Z}_2$ as a polynomial of the special functions $v_1, \ldots v_n$ together with the constant function 1 (which always returns 1).

5.2 Encoding

Now, consider how one might connect these Boolean functions to error-correcting codes. As was mentioned, any Boolean function can be represented as a polynomial of the functions $1, v_1, \ldots, v_n$. That polynomial has a coefficient vector $(a_1, \ldots, a_n)$, which we will suggestively call the message, and the output of the polynomial $c_1c_2 \ldots c_N$ which represents the codeword.

Definition 5.4. (Reed-Muller code) The rth order Reed-Muller code $\mathscr{R}(r, m)$ of length $n = 2^m$, where $0 \leq r \leq m$, is the set of all vectors f where $f(v_1, \ldots, v_m)$ is a Boolean function which is a polynomial of degree at most r.

In other words, the rth order Reed-Muller code of length 2^m is the set of all linear combinations of products of v_i's up to size r, i.e.

$$\sum_{\substack{S \subseteq [m] \\ |S| \leq r}} a_S \cdot \prod_{i \in S} v_i, \quad a_S \in \mathbb{Z}_2. \tag{5.1}$$

Hopefully a lengthy example will clarify this definition.

Example 5.5. The second order RM code of length 8, $\mathscr{R}(2, 3)$ consists of the 128 codewords

$$a_0\mathbf{1} + a_1v_1 + a_2v_2 + a_3v_3 + a_{12}v_1v_2 + a_{13}v_1v_3 + a_{23}v_2v_3, \quad a_i \in \mathbb{Z}_2.$$

This code has order 2 because each term in the polynomial is a product of at most two Boolean functions. By varying the coefficients a_i according to the bits of the message, we obtain a polynomial that corresponds to the codeword.

For example, suppose we wanted to encode the message $m = 0110101$. We would obtain the following function as the codeword

$$f_c = 0\mathbf{1} + 1v_1 + 1v_2 + 0v_3 + 1v_1v_2 + 0v_1v_3 + 1v_2v_3.$$

Now, performing addition and multiplication of the Boolean functions in the terms of f_c, we obtain:

$$\begin{aligned} f_c &= v_1 + v_2 + v_1v_2 + v_2v_3 \\ &= 01010101 \oplus 00110011 \oplus 00010001 \oplus 00000011 \\ &= 01110100. \end{aligned}$$

So, for message $m = 0110101$ the corresponding codeword is $c = 01110100$, which can also be thought of as the polynomial f_c above.

The set $\{\Pi_{i \in S} v_i \mid S \subseteq [m], |S| \leq r\} \cup \{\mathbf{1}\}$ forms a basis for the code, and it has size

$$\sum_{i=0}^{r} \binom{m}{i}$$

Theorem 5.6. $\mathscr{R}(r, m)$ has minimum distance 2^{m-r}.

Proof. This induction proof will be left as an exercise.

□

1	1111111111111111
v_4	0000000011111111
v_3	0000111100001111
v_2	0011001100110011
v_1	0101010101010101
v_3v_4	0000000000001111
v_2v_4	0000000000110011
v_1v_4	0000000001010101
v_2v_3	0000001100000011
v_1v_3	0000010100000101
v_1v_2	0001000100010001
$v_2v_3v_4$	0000000000000011
$v_1v_3v_4$	0000000000000101
$v_1v_2v_4$	0000000000010001
$v_1v_2v_3$	0000000100000001
$v_1v_2v_3v_4$	0000000000000001

Table 5.1: Generator matrix for $\mathscr{R}(4,4)$.

5.2.1 Generator Matrix

For those who prefer to think of encoding functions in terms of generator matrices, the generator matrix for $\mathscr{R}(r,m)$ has as its rows the terms of the polynomial in formula 5.1.

Theorem 5.7. The generator matrix G for $\mathscr{R}(r,m)$ has the following property

$$G(r,m) = \begin{bmatrix} G(r,m-1) & G(r,m-1) \\ 0 & G(r-1,m-1) \end{bmatrix}.$$

Example 5.8. The generator matrix for $\mathscr{R}(4,4)$ is displayed in Table 5.1. So, by simple matrix multiplication with our 16-bit message, we will obtain the corresponding codeword. Note that the generator matrix for $\mathscr{R}(3,4)$ is simply rows 1-15 of the above matrix, and the generator matrix for $\mathscr{R}(2,4)$ is just rows 1-11, and so on.

The rows of the above matrix form a basis for the code, meaning any codeword in $\mathscr{R}(4,4)$ can be expressed uniquely as a linear combination of the rows in the generator matrix. That unique linear combination is, in fact, the message that corresponds to the codeword. Now, we will look at the classical decoding algorithm.

5.3 Classical Decoding Algorithm

There are a few ways to approach the decoding algorithm for Reed-Muller codes. By far the most elegant is to view RM codes through the lens of finite geometries and utilize the framework that geometries provide. The reader is encouraged to read Appendix A before proceeding with the geometric description of the RM decoding algorithm.

5.3.1 Geometric Connections

Recall that the Euclidean geometry of dimension m over $GF(2)$, denoted $EG(m, 2)$, contains 2^m points whose coordinates are all of the binary vectors of length m, $(v_1, \ldots, v_m)$. If the zero point is deleted, the projective geometry $PG(m-1, 2)$ is obtained. We can think of the codewords of $\mathscr{R}(r, m)$ in this context, namely as subsets of $EG(m, 2)$. However, we must have a way to specify the subsets. This is achieved by looking at the codewords as *incidence vectors* for these subsets.

Definition 5.9 (Incidence vector)**.** Let $S \subseteq X$ where $|X| = 2^m$ and X is well-ordered. Then, $v \in \mathbb{Z}_2^m$ is an incidence vector for S if

$$v_i = 1 \iff x_i \in S,$$

where x_i is the i-th element of the ordered set X. Incidence vectors are also called *indicator vectors*.

Example 5.10. $EG(3, 2)$ consists of 8 points, call them $P_0, \ldots, P_7$, whose coordinates we may take to be the following column vectors:

	P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7
$\bar{v_1}$	1	0	1	0	1	0	1	0
$\bar{v_2}$	1	1	0	0	1	1	0	0
$\bar{v_3}$	1	1	1	1	0	0	0	0

The subset $S = \{P_2, P_3, P_4, P_5\}$ has incidence vector $\chi(S) = 00111100$. This is a codeword of $\mathscr{R}(1, 3)$. So, there is a one-to-one correspondence between codewords in $\mathscr{R}(1, 3)$ and subsets of $EG(3, 2)$. This property will prove useful for decoding.

For any value m, let us take the complements of the vectors $v_m, \ldots, v_1$.

	P_0	P_1	P_2	P_3	$\ldots$	P_{2^m-4}	P_{2^m-3}	P_{2^m-2}	P_{2^m-1}
$\bar{v}_m$	1	1	1	1	$\ldots$	0	0	0	0
$\bar{v}_{m-1}$	1	1	1	1	$\ldots$	0	0	0	0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\bar{v}_2$	1	1	0	0	$\ldots$	1	1	0	0
$\bar{v}_1$	1	0	1	0	$\ldots$	1	0	1	0

The columns of this matrix are the points in $EG(m, 2)$. Again, there is a one-to-one correspondence between the subsets of $EG(m, 2)$ and binary vectors of length 2^m. In particular, any vector $x \in \mathbb{Z}_2^m$ is an incidence vector for a subset of $EG(m, 2)$. The vectors v_i are themselves the characteristic vectors of hyperplanes which pass through the origin (subspaces of dimension $m - 1$), the $v_i v_j$ describe subspaces of dimension $m - 2$, and so on.

5.3.2 Decoding Algorithm

As with the other error-correcting codes we have studied thus far, RM decoding will rely on parity checks and *majority logic decoding*, which is a slightly different term for *maximum likelihood decoding*, or *nearest neighbor decoding*. Finite geometries will inform the algorithm as to exactly which parity checks to compute.

The decoding algorithm is a little involved, so we will describe the general decoding algorithm followed by an example of encoding and decoding. We will work with $\mathscr{R}(2, 4)$ to illustrate the algorithm notation rather than using general notation (recall that rows 1-11 of Table 5.1 contains the generator matrix for this code).

Denote the eleven bits of the message as $m = m_0 m_4 m_3 m_2 m_1 m_{34} m_{24} m_{14} m_{23} m_{13} m_{12}$. The notation here is taken to match the rows of Table 5.1. When we apply $m \cdot G$ where G is the generator matrix for $\mathscr{R}(2, 4)$, this message gets encoded into the following codeword.

$$\begin{aligned} m \cdot G &= m_0 1 + m_4 v_4 + \ldots + m_1 v_1 + \ldots + m_{12} v_1 v_2 \\ &\overset{call}{=} c_0 c_1 c_2 \ldots c_{15} \\ &= c. \end{aligned}$$

Now, the challenge is to decode $\hat{c}$, some noisy version of a codeword (note that $\hat{c}$ may or may not be a codeword itself). In general, to decode a noisy codeword from $\mathscr{R}(r, m)$, we construct a series of parity checks. First, we will find m_σ where $\sigma = \sigma_1 \ldots \sigma_r$, say. The corresponding row of the generator matrix $v_{\sigma_1} \ldots v_{\sigma_r}$ is the incidence vector of an $(m - r)$-dimensional subspace S of $EG(m, 2)$. To find S, use the row of the generator matrix as an indicator vector on the matrix of complements of the Reed-Muller monomial basis vectors. Now, we take T to be the complementary subspace to S with incidence vector $v_{\tau_1} \ldots v_{\tau_{m-r}}$ where $\{\tau_1, \ldots \tau_{m-r}\} = [m] \setminus \{\sigma_1, \ldots, \sigma_r\}$. Then T meets S at a single point, namely the origin. Now, let $U_1, \ldots, U_{2^{m-r}}$ be the *translates* of T in $EG(m, 2)$, including T itself. Each U_i meets S in exactly one point.

Theorem 5.11. If there are no errors, m_σ is given by

$$m_\sigma = \sum_{P \in U_i} c_P, \quad i = 1, \ldots, 2^{m-r}$$

Note that there are 2^{m-r} equations that should all yield the same m_σ, and if not, one can apply majority logic. This theorem implies that if no more than $\lfloor \frac{1}{2}(2^{m-r}-1) \rfloor$ errors occur, majority logic decoding will recover each of the symbols v_σ correctly, where σ is a string of any r symbols. After going through this process for all r-length subscripts in m, we subtract the corresponding $a_{ij}v_iv_j$'s from c to obtain a new codeword $c' = c'_0c'_1 \dots c'_{15}$ and repeat the process for all $r-1$-length subscripts. We continue this process until all we have left is $c_0 1 +$ error and $c_0 = 0$ or 1 according to the number of 1's in the remainder of the codeword.

5.3.3 Geometric Decoding Example

Again, we will work with $\mathscr{R}(2,4)$ since rows 1-11 of Table 5.1 are the generator matrix for the code. First, let us encode some message $m_0m_4m_3m_2m_1m_{34}m_{24}m_{14}m_{23}m_{13}m_{12} =$ 00001010100, which is three zero's followed by the ascii value for 'T' in binary ('T' for thesis).

Now, we encode this message by matrix multiplication (mod 2) to get $c_0 \dots c_{15} =$ 0101011001100101. Now, suppose the channel flips a bit in the process of sending the codeword, and $\hat{c} = 0101011\underline{1}01100101$ is what the decoder receives (without the flipped bit underlined, obviously).

The first bit of the message that we want to decode is m_{12}, so applying the notation used in the general decoding, we have $\sigma = 12$. Now, we look at the row corresponding to v_1v_2, which is row 11 of Table 5.1: 0001000100010001. This is our indicator vector for the complements of $v_1, \dots, v_4$, which is below:

$\bar{v_4}$	1111111100000000
$\bar{v_3}$	1111000011110000
$\bar{v_2}$	1100110011001100
$\bar{v_1}$	1010101010101010

So, from this table we get $S = \{1100, 1000, 0100, 0000\}$. Now, the incidence vector for T is the row of Table 5.1 corresponding to v_{34}. Thus, the incidence vector for T is 0000000001111. So, $T = \{0011, 0010, 0001, 0000\}$. Indeed $S \cap T = \{0000\}$.

Now, we take all translates of T in $EG(4,2)$ (that is, we take each point in S and translate T by that value). The translates of T are:

$$U_1 = \{1100 + t \mid t \in T\} = \{1111, 1110, 1101, 1100\}$$
$$U_2 = \{1000 + t \mid t \in T\} = \{1011, 1010, 1001, 1000\}$$
$$U_3 = \{0100 + t \mid t \in T\} = \{0111, 0110, 01010, 0100\}$$
$$U_4 = \{0000 + t \mid t \in T\} = \{0011, 0010, 0001, 0000\}$$

Now, we will use the bits of the received codeword $\hat{c}$ to compute m_{12}, the last bit of the message. By applying Theorem 5.11, we have that

$$m_{12} = \sum_{P \in U_i} c_P, \quad i = 1, 2, 3, 4$$

This gives us the four following checksums, which should all be equal in the case of no errors.

$$\begin{aligned} m_{12} &= c_{15} + c_{14} + c_{13} + c_{12} \\ &= c_{11} + c_{10} + c_9 + c_8 \\ &= c_7 + c_6 + c_5 + c_4 \\ &= c_3 + c_2 + c_1 + c_0 \end{aligned}$$

So, these equations give us that $m_{12} = 0, 0, 1, 0$. These equations are *not* equal because bit c_7 was flipped by the noisy channel. But, the decoder doesn't know which bit was flipped. Instead, it decides that $m_{12} = 0$ using majority logic. Indeed, this is correct! (Note how if two equations didn't match, the decoder would possibly make a decoding error).

Now, the decoder will continue using this process to decode m_{13}, m_{23}, m_{14}, and m_{34}. Once it has figured out those five bits, it will subtract $m_{34}v_3v_4 + m_{14}v_1v_4 + m_{23}v_2v_3 + m_{13}v_1v_3$ from $\hat{c}$ and proceed with the algorithm to decode $m_0 \dots m_4$. We will spare the reader these details, but that at least conveys the tedium involved in decoding. In the chapters that follow, we will explore a quantum way of decoding RM codes.

Chapter 6

Quantum Multivariate Polynomial Interpolation

Before we dive into quantum decoding algorithms for Reed-Muller codes, we must understand the mechanisms behind such an algorithm. The contents of this chapter are largely taken from the recent work [CITE MULTIVARIATE AND OTHER INTERPOLATION PAPER]. In this chapter, we set up the general algorithm that we will later tailor to our Reed-Muller decoding problem.

It is worth noting that the papers upon which this chapter is based ([CITE THEM]) is far more extensive than what will be covered here. In fact, their quantum polynomial interpolation algorithm is much more closely related to quantum cryptanalysis and a concept in cryptography called *secret sharing*. Alas, this is not a cryptography thesis.

6.1 The Problem Definition

Let $f(x_1, \ldots, x_n) \in \mathbb{F}_q[x_1, \ldots, x_n]$, that is, f is a multivariate polynomial over the finite field of size q, and let f be of degree d. Suppose an algorithm $\mathcal{A}$ has oracle access to f, and d is known. The polynomial interpolation problem that $\mathcal{A}$ must solve is to determine the coefficients of f by querying the oracle. In other words, f must learn the coefficient vector of f.

The learning theory question that researchers then ask is: *how many queries must any algorithm $\mathcal{A}$ make to the oracle before it can determine the coefficients of f?* In this case, researchers often care about some "worst case," also called a *lower bound*, over all possible algorithms $\mathcal{A}$ that solve the polynomial interpolation problem. Or, they may wonder what the optimal algorithm is.

Additionally, those interested in quantum complexity theory wonder how the an-

swer to the above question changes depending on whether $\mathcal{A}$ is a quantum or classical algorithm. The best classical algorithm for multivariate polynomial interpolation requires $\binom{n+d}{d}$ queries, using a well-chosen system of linear equations.

6.2 Quantum Query Model

Much like phase kickback, queries to the oracle will be encoded in the phase of the response register. However, the algorithm will perform queries in the Fourier basis using a technique that is a generalization of the phase kickback trick. Moreover, the algorithm will make k quantum queries in parallel to extract all of this information at once. These k queries are chosen *non-adaptively*, that is, the algorithm prepares the questions all at once.

6.2.1 Quantum Fourier Transform

Defining the quantum Fourier transform requires some background. First, it is a well-known fact that the order of any finite field can be written as a prime to some power. In other words, the finite field $\mathbb{F}_q$ has order $q = p^r$ where p is prime. Now, define the trace function $T : \mathbb{F}_q \to \mathbb{F}_q$ by $z \mapsto z + z^p + z^{p^2} + \ldots + z^{p^{r-1}}$. Next, we will define the exponential function $E : \mathbb{F}_q \to \mathbb{C}$ by $E(z) = e^{i2\pi T(z)/p}$. Now we are ready to define the quantum Fourier transform (QFT).

Definition 6.1 (QFT)**.** The quantum Fourier transform over $\mathbb{F}_q$ is a unitary transformation acting as

$$|x\rangle \longmapsto \frac{1}{\sqrt{q}} \sum_{y \in \mathbb{F}_q} E(xy)|y\rangle.$$

The k-dimensional quantum Fourier transform is given by

$$|x\rangle \longmapsto \frac{1}{q^{k/2}} \sum_{y \in \mathbb{F}_q^k} E(x \cdot y)|y\rangle.$$

This will be key to the phase query algorithm.

6.2.2 Phase Query

The authors of [CITE] describe what they call a *phase query*. Essentially, the algorithm involves doing an inverse quantum Fourier transform (which we will denote

QFT^{-1}), performing a query to oracle $\mathcal{O}$, and then doing a quantum Fourier transform (QFT) as follows

$$|x,y\rangle \xrightarrow{QFT^{-1}} \frac{1}{\sqrt{q}} \sum_{z\in\mathbb{F}_q} E(-yz)|x,z\rangle \tag{6.1}$$

$$\xrightarrow{\mathcal{O}} \frac{1}{\sqrt{q}} \sum_{z\in\mathbb{F}_q} E(-yz)|x,z+f(x)\rangle \tag{6.2}$$

$$\xrightarrow{QFT} \frac{1}{q} \sum_{z,w\in\mathbb{F}_q} E(-yz+(z+f(x))w)|x,w\rangle \tag{6.3}$$

$$= E(yf(x))|x,y\rangle. \tag{6.4}$$

So, overall the phase query has the following action $|x,y\rangle \mapsto E(yf(x))|x,y\rangle$. Now, instead of just querying f on one input $x \in \mathbb{F}_q$, we want to query it on k inputs. So, we choose a subset of $\mathbb{F}_q^k$ on which to perform k parallel queries, each in a separate register. For $x, y \in \mathbb{F}_q^k$, the k parallel phase queries has the following action

$$|x,y\rangle \longmapsto E\left(\sum_{i=1}^{k} y_i f(x_i)\right)|x,y\rangle$$

where x_i and y_i are the i-th bits of x and y, respectively.

6.3 More Preliminaries

Next, we define a mapping $Z : \mathbb{F}_q^{nk} \times \mathbb{F}_q^k \to \mathbb{F}_q^J$ by

$$Z(x,y)_j = \sum_{i=1}^{k} y_i x_i^j \quad \text{for } j \in \mathbb{J}$$

so that

$$Z(x,y)\cdot c = \sum_{i=1}^{k} y_i f(x_i)$$

where $\mathbb{J}$ is the set of allowed exponents, and $J \coloneqq |\mathbb{J}|$.

Now, we restrict the codomain of Z so that it forms a bijection. Let $R_k \coloneqq Z(\mathbb{F}_q^{nk}, \mathbb{F}_q^k)$ be the image of Z. Now, for each $z \in R_k$, choose a unique $(x,y) \in \mathbb{F}_q^{nk} \times \mathbb{F}_q^k$ such that $Z(x,y) = z$. Call T_k this set of unique representatives. Then, $Z : T_k \to R_k$ is a bijection.

6.4 The Algorithm

The algorithm has three steps. It starts in a superposition over T_k. First, it performs k phase queries (denoted k-$\mathcal{PQ}$), then it simply computes Z in place. Finally, we measure in the Fourier basis to get an approximation of the coefficients. More precisely,

$$\begin{aligned}\frac{1}{\sqrt{|T_k|}} \sum_{(x,y)\in T_k} |x,y\rangle &\xmapsto{k\text{-}\mathcal{PQ}} \frac{1}{\sqrt{|T_k|}} \sum_{(x,y)\in T_k} E(Z(x,y)\cdot c)|x,y\rangle \\ &\xmapsto{Z} \frac{1}{\sqrt{|R_k|}} \sum_{z\in R_k} E(z\cdot c)|z\rangle.\end{aligned}$$

Measuring in the basis of Fourier states defined as follows

$$|\hat{c}\rangle := \frac{1}{\sqrt{q^J}} \sum_{z\in\mathbb{F}_q^J} E(z\cdot c)|z\rangle,$$

where c are the computational basis states, results in the correct vector of coefficients with probability $|R_k|/q^J$.

Chapter 7

Quantum Decoding Algorithm for Reed-Muller Codes

In this chapter, we explore the possibility of using the quantum polynomial interpolation algorithm to decode Reed-Muller codes. The decoder will have oracle access to the received codeword f_c and, through a well-chosen series of queries and computations, will need to output the correct message m. The task of this chapter is to yet again reframe the Reed-Muller decoding algorithm. This time, instead of thinking about decoding in terms of finite geometries, we will explain it in terms of polynomial interpolation. Furthermore, the algorithm described in the previous chapter will be tailored to the task of decoding.

7.1 Polynomial : Coefficient Vector :: Codeword : Message

As was previously discussed, we can view codewords in $\mathscr{R}(r, m)$ as multivariate polynomials of degree at most r with coefficients in the field $\mathbb{F}_2$. The polynomials have the following form:

$$f_c = \sum_{\substack{S \subseteq [m] \\ |S| \leq r}} a_S \cdot \prod_{i \in S} v_i, \quad a_S \in \mathbb{Z}_2.$$

where f_c is the $\mathscr{R}(r, m)$ codeword that corresponds to message a. So, by applying the polynomial interpolation algorithm to f_c, we should be able to extract a.

7.1.1 Adapting Polynomial Interpolation

A lot of nice things happen when we adapt the polynomial interpolation algorithm to our use case. For one, the notation gets substantially less painful to look at. More importantly, many components of the algorithm become simpler.

First, the polynomials in $\mathscr{R}(r,m)$ are over the field $\mathbb{F}_2$. So, the order of the field is simply 2 and the trace function defined earlier $T : \mathbb{F}_2 \to \mathbb{F}_2$ defined by $z \mapsto z + z^p + z^{p^2} + \ldots + z^{p^{r-1}}$ is just the identity transformation. This simplifies the quantum Fourier transform. The function $E : \mathbb{F}_2 \to \mathbb{C}$ is now defined by $z \longmapsto e^{i\pi z}$.

Theorem 7.1. The quantum Fourier transform over $\mathbb{F}_2$ is just the Hadamard transform.

Proof. The QFT acts on a state $|x\rangle$, $x \in \mathbb{F}_2$, as follows

$$|x\rangle \longmapsto \frac{1}{\sqrt{2}} \sum_{y \in \mathbb{F}_2} E(xy)|y\rangle = \frac{1}{\sqrt{2}} \sum_{y \in \mathbb{F}_2} e^{i\pi xy}|y\rangle.$$

But, since $x, y \in \mathbb{F}_2$, this is just the transformation

$$\begin{aligned}
|x\rangle \longmapsto & \frac{1}{\sqrt{2}} \Big(E(x \cdot 0)|0\rangle + E(x \cdot 1)|1\rangle \Big) \\
= & \frac{1}{\sqrt{2}} \Big(e^{i\pi x \cdot 0}|0\rangle + e^{i\pi x \cdot 1}|1\rangle \Big) \\
= & \frac{1}{\sqrt{2}} \Big(1|0\rangle + e^{i\pi x}|1\rangle \Big) \\
= & \frac{1}{\sqrt{2}} \Big(1|0\rangle + (-1)^x|1\rangle \Big).
\end{aligned}$$

This is the Hadamard transform.

□

This simplifies the phase query. In the previous chapter we showed that the phase query has the following action $|x, y\rangle \mapsto E(yf(x))|x, y\rangle$. Given the simplified definition of QFT and E, we have that the phase query $(\mathcal{PQ})$ in $\mathbb{F}_2$ has the action

$$|x, y\rangle \xrightarrow{\mathcal{PQ}} (-1)^{yf(x)}|x, y\rangle. \tag{7.1}$$

Now, the only thing left to inspect is the mapping $Z : T_k \to R_k$ where $T_k \subseteq \mathbb{F}_2^{nk} \times \mathbb{F}_2^k$ and $R_k \subseteq \mathbb{F}_2^J$. In this mapping, the coordinate $(x, y) \in T_k$ is comprised of the query register (x) and response register (y). Each pair is a set of k queries to the oracle and

the corresponding results from the oracle of the k queries. In other words, we have that

$$(x, y) = \begin{bmatrix} x_1 & \mathcal{O}_f(x_1) \\ x_2 & \mathcal{O}_f(x_2) \\ \vdots & \vdots \\ x_k & \mathcal{O}_f(x_1) \end{bmatrix}.$$

Now, we may have that the response register y stores $r \oplus \mathcal{O}_f(x_i)$ where r is whatever was in the register prior to the query; these details are omitted because they yield equivalent definitions of Z.

The thing which Z computes is fundamentally difficult. The question that Z answers is: *Given k pairs of query inputs and query outputs, which polynomial is most likely?*. This problem is, in many ways, more difficult than plain polynomial interpolation. In order to even define Z, one must decide how it should behave given every possible k-set of input queries and all possible responses from the oracle. The one guarantee on Z is that it computes the results in place without any oracle access. Since this thesis is concerned with quantum query complexity, we may leave Z undefined with the understanding that the exact implementation of Z will not affect the query complexity of the algorithm.

7.1.2 The Algorithm

We begin in a superposition over T_k, make k parallel phase queries, compute Z in place, and then instead of measuring in the Fourier basis, we will take the Hadamard and measure in the computational basis. More precisely,

$$\begin{aligned} \frac{1}{\sqrt{|T_k|}} \sum_{(x,y)\in T_k} |x, y\rangle &\xrightarrow{k-\mathcal{PQ}} \frac{1}{\sqrt{|T_k|}} \sum_{(x,y)\in T_k} (-1)^{Z(x,y)\cdot c} |x, y\rangle \\ &\xrightarrow{Z} \frac{1}{\sqrt{|R_k|}} \sum_{z\in R_k} (-1)^{z\cdot c} |z\rangle \\ &\xrightarrow{H^{\otimes n}} \frac{1}{\sqrt{2^n|R_k|}} \sum_{z\in R_k} (-1)^{z\cdot c} \sum_{j\in\mathbb{Z}_2^n} (-1)^{z\cdot j} |j\rangle \\ &= \frac{1}{\sqrt{2^n|R_k|}} \sum_{z\in R_k} \sum_{j\in\mathbb{Z}_2^n} (-1)^{z(c+j)} |j\rangle. \end{aligned}$$

Now, measuring in the computational basis, we measure $|c\rangle$ with probability

$$\left(\frac{|R_k|}{\sqrt{2^n|R_k|}}\right)^2 = \left(\sqrt{\frac{|R_k|}{2^n}}\right)^2 = \frac{|R_k|}{2^n}$$

So, for k queries, $P_{succ} = |R_k|/2^n$ for Reed-Muller decoding.

7.2 Two Types of Noisy Oracles

One may wonder how the probability of success of the above algorithm is affected by an unreliable oracle, which is similar to the question that we answered about the binary simplex code. In the case of the simplex code, the algorithm only required one quantum query, but this algorithm makes k quantum queries. After some thought about noise models, two kinds of unreliable or noisy oracles arise: one which we will call stateful or self-consistent and one which we call stateless or contradictory.

First, we define these two models intuitively and later rigorously. Both oracles are unreliable or noisy in some way. The stateful oracle, when queried multiple times on a given input, will always output the same answer. One can imagine that the oracle *memoizes*[1] computations to all of the queries it has recieved and always checks the table to see if it was previously queried on the given input. In this way, it remains self-consistent.

The second model is the stateless, or contradictory, oracle. When the oracle is queried multiple times, it may give different answers to the same query. So, the query $\widehat{\mathcal{O}}_f(x)$ may yield $f(x)$ with some probability p and $\neg f(x)$ with probability $1-p$. We called this model *stateless* because the oracle cannot store the results of its previous computations and thus must recompute queries every time.

7.2.1 Connection to Message Transmission

In the context of error-correction, this relates to how the oracle is connected to the communication channel. If the noisy oracle is stateless, this is analogous to the message being transmitted once and stored in the oracle, with errors. As the oracle is queried, it simply consults the noisy codeword that it stored in memory somewhere rather than requesting the codeword to be resent on every query.

The stateless noisy oracle is analogous to the oracle requesting a retransmission of the codeword on every query. In other words, a new error vector is present on each query and this presumably comes from the noise in the communication channel as the codeword is retransmitted.

[1] That is, it saves its answers.

Conclusion

Many questions arose during the course of the year. Some of them yielded satisfactory answers, while others turned up disappointing results and were not included. Yet more remained unanswered. The project of this thesis was to: 1) shed clarity on these confusing topics, and in doing so, 2) inspire further work in this area. Perhaps one day the topics that were left unpursued can be realized concretely. They are described below.

8.1 Further Work

In this thesis we presented the idea of algorithmic robustness to noisy quantum oracles. That is, for a given quantum algorithm $\mathcal{A}$ which relies on quantum queries to oracle $\mathcal{O}$, what is P_{succ} of the algorithm if the oracle is replaced with a noisy oracle $\widehat{\mathcal{O}}$?

We were able to answer this question for the binary simplex decoding algorithm if we had a specific known error rate ($k/2^n$) for the noisy oracle. This topic could be generalized by considering the probability of success given that the noise added to the oracle *follows some particular distribution*. This is more realistic because it is unlikely that one would know the exact number of errors that occurred in the query process.

A second way to expand upon this thesis is to answer questions about robustness to noisy oracles of a *wide variety* of algorithms. For example, Grover's search algorithm [Grover, 1996] seems particularly suited for this line of inquiry.

Lastly, for an algorithm which relies on *multiple* oracle queries, one could analyze the difference in P_{succ} when the oracle is replaced with a *stateless* versus a *stateful* noisy oracle (as described in chapter 7).

Appendix A

Finite Geometries

Finite geometries are mathematical objects that codes map onto well, and they provide a new way of viewing codes. In this appendix, we will explore the properties of finite geometries that are vital to the classical Reed-Muller decoding algorithm with a focus on the projective geometry and affine or Euclidean geometry. This material is largely based on Appendix B in [REFERENCE MACWILLIAMS AND SLOANE]

Definition A.1. (Projective geometry) A finite projective geometry consists of a finite set Ω of *points* $p, q, \ldots$ together with a collection of subsets $L, M, \ldots$ of Ω called *lines*, which satisfies the following axioms (If $p \in L$ we say that p lies on L or L passes through p.)

1. There is a unique line (denoted by (pq)) passing through any two distinct points p and q.

2. Every line contains at least 3 points.

3. If distinct lines L, M have a common point p, and if $q, r \neq p$ are points of L and $s, t \neq p$ are points of M, then the lines (qt) and (rs) also have a common point.

4. For any point p there are at least two lines not containing p, and for any line L there are at least two points not on L.

A *subspace* of the projective geometry is a subset S of Ω such that

5. If p, q are distinct points of S, then S contains all of the points of (pq)

A *hyperplane* H is a maximal proper subspace, so that Ω is the only subspace which properly contains H.

Definition A.2 (Euclidean geometry)**.** An affine or Euclidean geometry is obtained by deleting the points of a fixed hyperplane H (called the hyperplane at infinity) from the subspaces of a projective geometry. The resulting sets are called the subspaces of the affine geometry.

A set T of points in a projective or affine geometry is called *independent* if, for every $x \in T, x$ does not belong to the smallest subspace which contains $T/\{x\}$. The *dimension* of a subspace S is $r-1$, where r is the size of the largest set of independent points in S. In particular, if $S = \Omega$ this defines the dimension of the projective geometry.

We denote Euclidean and projective geometries of dimension m constructed from a finite field $GF(q)$ by $EG(m, q)$ and $PG(m, q)$, respectively. Now, we will discuss the projective and affine geometries that are obtained from finite fields, as they are most pertinent to Reed-Muller codes.

Let $GF(q)$ be a finite field and suppose $m \geq 2$. We will take the points of Ω to be the nonzero $(m+1)$-tuples in $GF(q)^{m+1}$ such that

$$(a_0, \ldots, a_m) \equiv (\lambda a_0, \ldots \lambda a_m) \quad \lambda, a_i \in GF(q), \quad \lambda \neq 0.$$

These are called *homogenous coordinates* for the points.

A *hyperplane* of subspace of dimension $m-1$ in $PG(m, q)$ consists of those points $(a_0, \ldots, a_m)$ which satisfy a homogenous linear equation

$$\lambda_0 a_0 + \lambda_1 a_1 + \ldots + \lambda_m a_m = 0, \quad \lambda_i \in GF(q)$$

The affine or projective geometry $EG(m, q)$ is obtained from $PG(m, q)$ by deleting the points of any hyperplane H. A subspace of $EG(m, q)$ of dimension r is called an *r-flat*.

Bibliography

Andris Ambainis. Understanding quantum algorithms via query complexity. *arXiv preprint arXiv:1712.06349*, 2017.

Dana Angluin. Queries and concept learning. *Machine learning*, 2(4):319–342, 1988.

Dana Angluin. Queries revisited. *Theoretical Computer Science*, 313(2):175–194, 2004.

Alp Atici and Rocco A Servedio. Improved bounds on quantum learning algorithms. *Quantum Information Processing*, 4(5):355–386, 2005.

Alexander Barg and Shiyu Zhou. A quantum decoding algorithm of the simplex code. In *PROCEEDINGS OF THE ANNUAL ALLERTON CONFERENCE ON COMMUNICATION CONTROL AND COMPUTING*, volume 36, pages 359–365. UNIVERSITY OF ILLINOIS, 1998.

Jianxin Chen, Andrew M Childs, and Shih-Han Hung. Quantum algorithm for multivariate polynomial interpolation. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 474(2209):20170480, 2018.

Andrew W Cross, Graeme Smith, and John A Smolin. Quantum learning robust against noise. *Physical Review A*, 92(1):012327, 2015.

Ronald De Wolf et al. *Quantum computing and communication complexity*. Citeseer, 2001.

Richard P Feynman. Simulating physics with computers. *International journal of theoretical physics*, 21(6):467–488, 1982.

Stephen Gortler and Rocco Servedio. Quantum versus classical learnability. In *Proceedings of the 16th Annual Conference on Computational Complexity*, pages 138–148, 2001.

Lov K Grover. A fast quantum mechanical algorithm for database search. *arXiv preprint quant-ph/9605043*, 1996.

Florence Jessie MacWilliams and Neil James Alexander Sloane. *The theory of error-correcting codes*, volume 16. Elsevier, 1977.

Michael A Nielsen and Isaac Chuang. Quantum computation and quantum information, 2002.

Peter W Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review*, 41(2):303–332, 1999.

Wim Van Dam. Quantum oracle interrogation: Getting all information for almost half the price. In *Proceedings 39th Annual Symposium on Foundations of Computer Science (Cat. No. 98CB36280)*, pages 362–367. IEEE, 1998.

Leiven M.K. Vandersypen, Matthias Steffen, Gregory Breyta, Costantino S Yannoni, Mark H Sherwood, and Isaac L. Chuang. Experimental realization of shor's quantum factoring algorithm using nuclear magnetic resonance. *Nature*, 2001.

Index

qubit, 2