Stateful and Stateless Noisy Quantum Oracles

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 Louise Swanberg

May 2019

Approved for the Division
(Mathematics)

James D. Pommersheim

Adam Groce

Preface

Theorem 0.1. a

Definition 0.2. b

Example 0.3. c

List of Abbreviations

You can always change the way your abbreviations are formatted. Play around with it yourself, use tables, or come to CUS if you'd like to change the way it looks. You can also completely remove this chapter if you have no need for a list of abbreviations. Here is an example of what this could look like:

ABC	American Broadcasting Company
CBS	Columbia Broadcasting System
CDC	Center for Disease Control
CIA	Central Intelligence Agency
CLBR	Center for Life Beyond Reed
CUS	Computer User Services
FBI	Federal Bureau of Investigation
NBC	National Broadcasting Corporation

Contents

Chapter 1: Introduction to 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 Phase 6
- 1.7 Universal Quantum Gate 7
- 1.8 No-Cloning Theorem 8
- 1.9 Quantum Oracles 9
  - 1.9.1 Quantum Queries 9
- 1.10 Phase-kickback Trick 10
- 1.11 Bernstein-Vazirani Algorithm 10
- 1.12 A Note on Asymptotic Complexity 11

Chapter 2: Introduction to Code Theory and Error-Correcting Codes **13**
- 2.1 Preliminaries 14
- 2.2 Overview of Transmission 14
- 2.3 Encoding 15
- 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: Introduction to Learning Theory **19**
- 3.1 Basics of Concept Learning 19
- 3.2 Certificate Complexity 19
- 3.3 Block Sensitivity 19

Chapter 4: Simplex Code **21**
4.1 Encoding 21
4.1.1 Example 22
4.2 Quantum Decoding Algorithm 22
4.3 Rosbustness to Errors Metric 22

Chapter 5: Reed-Muller Codes **23**
5.1 Boolean Functions 23
5.2 Encoding 23
5.3 Classical Decoding Algorithm 23

Chapter 6: Quantum Decoding Algorithm for Reed-Muller Codes . . **25**
6.1 Multivariate Polynomial Interpolation 25
6.2 Stateful Noisy Oracle 25
6.3 Stateless Noisy Oracle 25

Conclusion **27**
6.1 More info 27

Appendix A: Asymptotic Complexity **29**

Appendix B: Tensors **31**

Appendix C: Finite Geometries **33**

Appendix D: TODO List **35**
D.1 Chapter 1: Quantum 35
D.2 Chapter 2: Coding Theory 35
D.3 Chapter 3: Learning Theory 35
D.4 Chapter 4: Simplex Code 35
D.5 Chapter 5: Reed-Muller Codes 35
D.6 Chapter 6: Quantum Decoding for RM Codes 35
D.7 Appendices 36
D.7.1 Asymptotic Complexity 36
D.7.2 Tensors 36
D.7.3 Finite Geometries 36

Bibliography **37**

Chapter 1

Introduction to 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 tasks, such as searching an unstructured list, but they have the same asymptotic runtime[1] as classical computers for other tasks. 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] We will discuss what this means later

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*[2]. 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. We can also think of $|\psi\rangle$ as a vector in a two-dimensional complex vector space, say $\mathbb{C}^2$. The states $|0\rangle$ and $|1\rangle$ form an orthonomal basis in this complex vector space and are called the *computational basis states.*

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 $|\beta|^2$. In other words, $|\psi\rangle$ is like a weighted coin that we can flip once to get either heads or tails, but we have no way to directly measure the bias in the coin. Unlike a coin, 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. In the coin analogy, it's like we have a weighted coin that we can flip once to reveal heads or tails, and every subsequent flip always yields the initial state that we flipped to.

You may be wondering why this is useful. 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.

---

[2] See reference on notation–not yet written

1.2 Quantum Logic Gates

As we already glossed over, $|0\rangle = 1|0\rangle + 0|1\rangle$ 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 $X|\psi\rangle = \beta|0\rangle + \alpha|1\rangle$. Alternatively, 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

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

Now, applying the transformation to our basis vectors, we get $\widehat{H}|0\rangle = (|0\rangle + |1\rangle)/\sqrt{2}$ and $\widehat{H}|1\rangle = (|0\rangle + |1\rangle)/\sqrt{2}$. There is still a problem with this transformation. All quantum transformations must be *reversible*, but we have mapped the two basis vectors to the same state, so the transformation is not reversible. To fix this, we will simply change the image of this transformation under $|1\rangle$:

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

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 to work in quantum algorithms. It 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 1.1 (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

To represent multiple qubits, we expand our two quantum states to many using tensor products [3]. For example, in a two qubit system, we have the following basis states:

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

These four states can equivalently be written as:

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

---

[3] See appendix on tensor products–not yet written

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

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

where i is the binary representation of the numbers $0, \ldots, n-1$. 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:

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

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 the equation above). 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. In particular, the n-qubit Hadamard transform is $H^{\otimes n} = H \otimes H \otimes \ldots \otimes H$, i.e. n tensor products.

1.5 Hadamard Transform

As promised, a more precise definition of the Hadamard transform follows. This is by far the most important unitary transformation in the field of quantum algorithms.

Definition 1.2 (Hadamard Transform)**.** The n-qubit Hadamard transform, $H^{\otimes n}$ acts on a state $|\psi\rangle$ as follows:

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

where ψ and j are n-bit binary numbers.

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

$$\begin{aligned} H^{\otimes n}|0\rangle^{\otimes n} &= |+\rangle^{\otimes n} \\ H^{\otimes n}|1\rangle^{\otimes n} &= |-\rangle^{\otimes n} \end{aligned}$$

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

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

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 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 $e^{i\theta}|\psi\rangle$,[4] where $|\psi\rangle$ is a state vector and θ is a real number. These two phases are equivalent up to their *global phase shift.* As a thought experiment, one may wonder what happens upon measuring the two states. Suppose M_x is our measurement operator. Then, the probabilities for x occurring upon measurement are: $\langle\psi|M_x^\dagger M_x|\psi\rangle$ and $\langle\psi|e^{-i\theta}M_x^\dagger M_x e^{i\theta}|\psi\rangle = \langle\psi|M_x^\dagger M_x|\psi\rangle$. So, the two states are indistinguishable with respect to measurements.

---

[4] HOW TO FORMAT EULERS NUMBER

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 the same *magnitude* on their amplitudes (i.e. $1/\sqrt{2}$) on both $|0\rangle$ and $|1\rangle$. Thus, they have the same *relative phase*. However, we can distinguish between the states $|+\rangle$ and $|-\rangle$ if we apply a 45 degree clockwise rotation to the states, we may distinguish between the states with probability 1 upon measurement. So, despite having the same relative phase, the states are, in fact, computationally distinct.

1.7 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 one universal quantum gate, however the following three gates are enough to make an *arbitrary approximation* of any quantum gate: Hadamard, CNOT, and phase gate.

The phase gate is defined

$$\begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix} \tag{1.2}$$

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.

$$\text{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{1.3}$$

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.

1.8 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

$$|\psi\rangle \otimes |s\rangle \xrightarrow{U} U(|\psi\rangle \otimes |s\rangle) = |\psi\rangle \otimes |\psi\rangle \tag{1.4}$$

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

$$U(|\psi\rangle \otimes |s\rangle) = |\psi\rangle \otimes |\psi\rangle \tag{1.5}$$

and

$$U(|\phi\rangle \otimes |s\rangle) = |\phi\rangle \otimes |\phi\rangle \tag{1.6}$$

Now, taking the inner product of these two equations gives[5]

$$\begin{aligned}
(\langle\psi| \otimes \langle s|)U^\dagger U|\phi\rangle \otimes |s\rangle &= \langle\psi \mid \phi\rangle \otimes \langle s \mid s\rangle \otimes I && (1.7)\\
&= \langle\psi \mid \phi\rangle && (1.8)\\
&= ((\langle\psi| \otimes \langle\psi|)(|\phi\rangle \otimes |\phi\rangle)) && (1.9)\\
&= \langle\psi \mid \phi\rangle \otimes \langle\psi \mid \phi\rangle && (1.10)\\
&= (\langle\psi \mid \phi\rangle)^2 && (1.11)
\end{aligned}$$

Line (5) holds because s is normalized. Critically, lines (5) and (8) 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 states which we know nothing about.

---

[5] Check this math???

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 states, and much of classical computation relies upon this fact.

1.9 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.

1.9.1 Quantum Queries

First, consider the action of the query. In order to ensure the reversibility of quantum queries, 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. This register remains untouched by the oracle. The response register has some initial state $|r\rangle$ and after the quantum query takes place, has the state $|r \oplus f(x)\rangle$ where f is the function that the oracle computed.

Definition 1.3 (Quantum Oracle Query)**.** A quantum oracle 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 have a superposition of queries of length n

$$|x\rangle = \sum_{i \in \mathbb{Z}_2^n} \alpha_i |i\rangle$$

and n response registers $|r\rangle = |r_1\rangle \otimes \dots |r_n\rangle$. Then, the query $\mathcal{O}(|x, r\rangle)$ results in the following state

$$|x\rangle = \sum_{i \in \mathbb{Z}_2^n} \alpha_i |i, r_i \oplus f(i)\rangle.$$

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

1.10 Phase-kickback Trick

Suppose we have "black-box access" to some function f. That is, we can query f on some input x and it will compute $f(x)$. We record the output in a *response register*, the second qubit in the example below; the first qubit is called the *query register*. This can be modeled as the following unitary transformation:

$$|x, z\rangle \xrightarrow{f} |x, z \oplus f(x)\rangle$$

Given z, we can deduce what $f(x)$ is by taking $z \oplus f(x) \oplus z = f(x)$, addition mod 2. This is essential to keeping the transformation unitary.

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.

1.11 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.

[CITATION TO VAZIRANI PAPER] The Bernstein-Vazirani algorithm solves the following problem: for $N = 2^n$, we are given $x \in \{0, 1\}^N$ with the property that there exists some unknown $a \in \{0, 1\}^n$ such that $x_i = i \cdot a \bmod 2$. The goal is to find a. In other words, x_i is the ith bit of the binary representation of x.

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 the bits of x in the phase of our state. Then, another Hadamard transform on all n qubits. With a simple measurement, we obtain a with probability 1. Now, for the details.

$$
\begin{aligned}
|0\rangle^{\otimes n} &\xrightarrow{H^{\otimes n}} \frac{1}{\sqrt{2^n}} \sum_{i\in\{0,1\}^n} |i\rangle && (1.12)\\
&\xrightarrow{\mathcal{O}_x} \frac{1}{\sqrt{2^n}} \sum_{i\in\{0,1\}^n} (-1)^{x_i}|i\rangle && (1.13)\\
&\xrightarrow{H^{\otimes n}} \frac{1}{2^n} \sum_{i\in\{0,1\}^n} (-1)^{x_i} \sum_{j\in\{0,1\}^n} (-1)^{i\cdot j}|j\rangle && (1.14)\\
&= |a\rangle && (1.15)
\end{aligned}
$$

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 $(-1)^{x_i} = (-1)^{(i\cdot a) \bmod 2} = (-1)^{(i\cdot a)}$, given by $x_i = (i \cdot a) \bmod 2$ in the problem statement. Thus, we may manipulate 1.14 as follows:

$$
\begin{aligned}
\frac{1}{2^n} \sum_{i\in\{0,1\}^n} (-1)^{x_i} \sum_{j\in\{0,1\}^n} (-1)^{i\cdot j}|j\rangle &= \frac{1}{2^n} \sum_{i\in\{0,1\}^n} (-1)^{(i\cdot a)} \sum_{j\in\{0,1\}^n} (-1)^{i\cdot j}|j\rangle && (1.16)\\
&= \frac{1}{2^n} \sum_{j\in\{0,1\}^n} \sum_{i\in\{0,1\}^n} (-1)^{i\cdot(a+j)}|j\rangle && (1.17)
\end{aligned}
$$

Now, note that $(-1)^{i\cdot(a+j)}$ is equal to 1 if and only if $j = a$. Thus, the inner summation evaluates to $1/2^n$ when $j = a$ and zero otherwise. So, 1.17 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.

1.12 A Note on Asymptotic Complexity

In this thesis, we will be studying the asymptotic complexity of various different metrics, including: runtime, query complexity, certificate complexity, block sensitivity, and perhaps circuit complexity[6]. Those who are unfamiliar with asymptotic complexity (also termed Landau-Bachmann notation in mathematics) should review the appendix on the topic.

---

[6] These complexity metrics will be defined later.

Chapter 2

Introduction to Code Theory and 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, 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.*

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 $(n-k) \times n$ 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 $e = c_1 \ldots e_n$ to the codeword so that the decoder receives a "noisy codeword" or "received vector" $r = c + m \pmod 2$.

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 $x = x_1 \ldots x_n$ 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:

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

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

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

So, now we want to maximize the equation

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

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 $dist(r, c)$ In other words, $c = \min_{c' \in \mathscr{C}}\{dist(c', r)\}$. 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 at most $\lfloor \frac{1}{2}(d-1) \rfloor$ errors.

Proof. Suppose we have a linear code $\mathscr{C}$ with minimum distance d. Given a received codeword $r = c + e$, nearest neighbor decoding will return c if and only if $dist(c, r) = \min_{c' \in \mathscr{C}}\{dist(c', r)\}$, i.e. if c is the only codeword that r is close to.

Then, for any given received vector $r = e + c$ the nearest neighbor decoding algorithm will output the correct c with probability 1 as long as $dist(c, r) = \min_{c' \in \mathscr{C}}\{dist(c', r)\}$ $= \min_{c' \in \mathscr{C}}\{dist(c', e + c)\} = \min_{c' \in \mathscr{C}}\{dist(c', c) + dist(c', e)\} < d$. □

2.4.4 Decoding Errors

other: cosets, syndrome(?), Perr Error vector. most lik Error-detection vs. error correction.

Chapter 3

Introduction to Learning Theory

Read Ronald de Wolf's thesis section on query complexity! pg. 193

3.1 Basics of Concept Learning

3.2 Certificate Complexity

3.3 Block Sensitivity

Chapter 4

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 compute the i-th bit of the codeword for a message m, take $c_i = i_2 \cdot m \bmod 2$, where i_2 is the binary representation of the number i. Thus, a codeword c for a message m is defined:

$$c_1 \dots c_{2^n} = 1_2 \cdot m || 2_2 \cdot m || \dots || (2^n)_2 \cdot m$$

computing all of the inner products modulo 2.

4.1.1 Example

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.

4.2 Quantum Decoding Algorithm

The quantum decoding algorithm for the simplex code is very pleasing. The first key observation is that the generating matrix for the simplex code is, in a sense, “equivalent” to the Hadamard matrix.

4.3 Rosbustness to Errors Metric

Chapter 5

Reed-Muller Codes

5.1 Boolean Functions

5.2 Encoding

5.3 Classical Decoding Algorithm

Projective geometry description

Chapter 6

Quantum Decoding Algorithm for Reed-Muller Codes

6.1 Multivariate Polynomial Interpolation

6.2 Stateful Noisy Oracle

6.3 Stateless Noisy Oracle

Conclusion

Here's a conclusion, demonstrating the use of all that manual incrementing and table of contents adding that has to happen if you use the starred form of the chapter command. The deal is, the chapter command in LaTeX does a lot of things: it increments the chapter counter, it resets the section counter to zero, it puts the name of the chapter into the table of contents and the running headers, and probably some other stuff.

So, if you remove all that stuff because you don't like it to say "Chapter 4: Conclusion", then you have to manually add all the things LaTeX would normally do for you. Maybe someday we'll write a new chapter macro that doesn't add "Chapter X" to the beginning of every chapter title.

6.1 More info

And here's some other random info: the first paragraph after a chapter title or section head *shouldn't be* indented, because indents are to tell the reader that you're starting a new paragraph. Since that's obvious after a chapter or section title, proper typesetting doesn't add an indent there.

Appendix A

Asymptotic Complexity

Computer scientists are very interested in how long their algorithm or function will take to run on given inputs. The crudest, most theoretical, runtime analysis looks at the *asymptotic runtime* of an algorithm. In practice, the actual runtime can be affected significantly by any number of factors like the operating system and memory latency, but for our purposes the *asymptotic* runtime is the only metric we will care about (particularly because current existent quantum computers are rather rudimentary).

The basic idea behind asymptotic runtime analysis is that we consider how long our algorithm takes to compute as a function of the *length* (number of bits) of the input, where the time interval is calculated as the number of basic operations required. Classically, the basic operations are generally considered to be the ones that the Arithmetic Logic Unit (ALU) in the CPU can do. These include: addition, subtraction, multiplication, division, boolean comparisons, and variable assignment. Quantumly, the basic operations are the same as above in addition to arbitrary unitary transformations and oracle queries (which take one time step).

Now that we have the basic idea, time to get technical. We define notions of relative asymptotic growth rates of functions: O, Ω, o, ω, and Θ. We will define these notions rigorously below, but for all practical purposes, the following distinctions are sufficient. Suppose we have two functions $f(n)$ and $g(n)$. Then,

- We say $f(n) = O(g(n))$, *"f is big-oh of g,"* if $f(n) \leq g(n)$ as n gets large;
- $f(n) = o(g(n))$, *"f is little-oh of g,"* if $f(n) < g(n)$ (strictly) as n gets large;
- $f(n) = \Omega(g(n))$, *"f is big-omega of g,"* if $f(n) \geq g(n)$ as $n \to \infty$;
- $f(n) = \omega(g(n))$, *"f is little-omega of g,"* if $f(n) > g(n)$ (strictly) as n gets large.

- Lastly, $f(n) = \Theta(g(n))$, *"f is theta of g,"* if $f(n) = g(n)$ (up to constant factors) as n gets large.

The above imprecise definitions are rigorous enough for our discussions; however, I present the formal definitions below for the overly pedantic or curious reader.

Theorem A.1. A function f(n) is O(g(n)), denoted f(n) = O(g(n)), if there exist $c > 0, n_0 > 0$ such that

$$f(n) \leq cg(n) \quad \forall n \geq n_0.$$

Theorem A.2. A function f(n) is o(g(n)), denoted f(n) = o(g(n)), if there exists $n_0 > 0$ such that for all $c > 0$,

$$f(n) < cg(n) \quad \forall n \geq n_0.$$

Theorem A.3. A function f(n) is $\Omega(g(n))$, denoted $f(n) = \Omega(g(n))$, if there exist $c > 0, n_0 > 0$ such that

$$f(n) \geq cg(n) \quad \forall n \geq n_0.$$

Theorem A.4. A function f(n) is $\omega(g(n))$, denoted $f(n) = \omega(g(n))$, if there exists $n_0 > 0$ such that for all $c > 0$,

$$f(n) > cg(n) \quad \forall n \geq n_0.$$

Theorem A.5. A function f(n) is $\Theta(g(n))$, denoted $f(n) = \Theta(g(n))$, if f(n) = O(g(n)) and g(n) = O(f(n)).

Determining the exact values of n_0 and c is not necessary. Instead, we will present a few heuristics:

1. Constant factors don't matter.

2. Only the largest complexity class matters if we are adding terms.

For example, $f(n) = 25n^3 + 170000n$ is $O(n^{17})$ because $n^3 \leq n^{17}$ as n grows, and $f(n) = \Theta(n^3)$ because the $170000n$ doesn't matter as n grows and we don't care about the constant factor of 25 either, and $f(n) = \omega(n^2)$, because $n^2 < n^3$ as n goes to infinity.

Appendix B

Tensors

Appendix C

Finite Geometries

Appendix D

TODO List

D.1 Chapter 1: Quantum

Finish section on phase

D.2 Chapter 2: Coding Theory

Write this section

D.3 Chapter 3: Learning Theory

Write this section Include original proofs

D.4 Chapter 4: Simplex Code

Write this section Include cool formula/proof :)

D.5 Chapter 5: Reed-Muller Codes

Write this section (maybe write after section on finite geometries)

D.6 Chapter 6: Quantum Decoding for RM Codes

Write this section

D.7 Appendices

D.7.1 Asymptotic Complexity

Write this to be more general for runtime and query complexity.

D.7.2 Tensors

Write this appendix

D.7.3 Finite Geometries

Write this appendix

Bibliography

Angel, E. (2000). *Interactive Computer Graphics : A Top-Down Approach with OpenGL*. Boston, MA: Addison Wesley Longman.

Angel, E. (2001a). *Batch-file Computer Graphics : A Bottom-Up Approach with QuickTime*. Boston, MA: Wesley Addison Longman.

Angel, E. (2001b). *test second book by angel*. Boston, MA: Wesley Addison Longman.

Deussen, O., & Strothotte, T. (2000). Computer-generated pen-and-ink illustration of trees. *"Proceedings of" SIGGRAPH 2000*, (pp. 13–18).

Fisher, R., Perkins, S., Walker, A., & Wolfart, E. (1997). *Hypermedia Image Processing Reference*. New York, NY: John Wiley & Sons.

Gooch, B., & Gooch, A. (2001a). *Non-Photorealistic Rendering*. Natick, Massachusetts: A K Peters.

Gooch, B., & Gooch, A. (2001b). *Test second book by gooches*. Natick, Massachusetts: A K Peters.

Hertzmann, A., & Zorin, D. (2000). Illustrating smooth surfaces. *Proceedings of SIGGRAPH 2000*, *5*(17), 517–526.

Jain, A. K. (1989). *Fundamentals of Digital Image Processing*. Englewood Cliffs, New Jersey: Prentice-Hall.

Molina, S. T., & Borkovec, T. D. (1994). The Penn State worry questionnaire: Psychometric properties and associated characteristics. In G. C. L. Davey, & F. Tallis (Eds.), *Worrying: Perspectives on theory, assessment and treatment*, (pp. 265–283). New York: Wiley.

Noble, S. G. (2002). *Turning images into simple line-art*. Undergraduate thesis, Reed College.

Reed College (2007). Latex your document. `http://web.reed.edu/cis/help/LaTeX/index.html`

Russ, J. C. (1995). *The Image Processing Handbook, Second Edition*. Boca Raton, Florida: CRC Press.

Salisbury, M. P., Wong, M. T., Hughes, J. F., & Salesin, D. H. (1997). Orientable textures for image-based pen-and-ink illustration. *"Proceedings of" SIGGRAPH 97*, (pp. 401–406).

Savitch, W. (2001). *JAVA: An Introduction to Computer Science & Programming*. Upper Saddle River, New Jersey: Prentice Hall.

Wong, E. (1999). *Artistic Rendering of Portrait Photographs*. Master's thesis, Cornell University.