On Quantum Computation

by

Fedor Labounko

A Thesis submitted to the Faculty
in partial fulfillment
of the requirements for the
BACHELOR OF ARTS

Accepted

_____
Allen Altman, Thesis Advisor

_____
Michael Bergman, Second Reader

_____
Paul Shields, Third Reader

_____
Mary B. Marcy, Provost and Vice President

Simon's Rock College
Great Barrington, Massachusetts
2006

Abstract

On Quantum Computation

by

Fedor Labounko

Simon's Rock College

Allen Altman, Thesis Advisor

This paper aims to introduce the reader to quantum computation and to implement quantum algorithms to find generators for an abelian group. We present a brief introduction to the necessary linear algebra, and define the postulates needed for quantum computing. Common quantum computing operators are introduced and then used in a presentation of Grover's search and Shor's factoring algorithms. We present some fundamental results on limitations as well as strengths of quantum computing involving the No-Cloning and No-Distinguishability theorems and circuits demonstrating quantum teleportation and superdense coding. We follow with a brief history of quantum complexity theory, citing results which question the Strong Church-Turing thesis as well as results which showcase the power of quantum computers.

The paper concludes with results on finding the generators of an abelian group. Three algorithms are presented which treat the case of a finite abelian group, of which two employ the efficient quantum solution to Simon's Hidden subgroup problem. An algorithm is presented for the case of the torsion-free abelian group and partial results on its running time analysis are discussed. We conclude with a brief review of the merits of quantum computation.

Acknowledgements

I owe a great deal to professors, friends, and members of my family, all too numerous to list fully, for help in writing this thesis and for their support and suggestions throughout the process. I would like to thank my thesis committee members Michael Bergman and Paul Shields for their invaluable comments and advice; in no small part thanks to them the subject matter is far more comprehensible than it would be otherwise. I am grateful to Mary, Daniel, and Caroline for providing balance in my life and keeping me in touch with sanity. Without you guys, I would have fallen apart. I am forever indebted to my parents who have always stood by me with sound advice, encouragement, and moral support. Without them, I would never be where I am today. Most thanks, however, should go to Allen for not only tirelessly working with me throughout the whole process, but also for providing me with an endless supply of encouragement, counsel, and inspiration. One could not possibly hope for a better advisor and friend.

The results presented in Chapter 6 were developed jointly with John Basias, Alex Eustis, and Ashley Kent at the 2004 summer REU at SUNY Potsdam and under the supervision of professor Kazem Mahdavi. I have learned much from working with them and owe them a debt of gratitude. Last, but certainly not least, I would like to thank the Simon's Rock Community for being my home for these past four years. The things I have learned and experiences I have had will remain with me for the rest of my life. Thank you.

Contents

Chapter 1

Fundamentals

1.1 Introduction

Quantum computation is the study of the information processing that can be done on a system operating on quantum particles. The introduction of quantum mechanical phenomena to the computational model produces a structure quite different from the traditional Turing machine model. A Turing machine is a finite state machine with a finite alphabet of symbols and an infinite tape of memory. The *Church-Turing thesis* is the conjecture that anything we might consider computable is computable by a Turing machine. This conjecture has become widely accepted by researchers in the field and is not challenged by the introduction of quantum phenomena. Another conjecture driven by practical results, however, concerns the issue of efficient computation. There is no known efficient algorithm, for instance, enabling Turing machines to factor large numbers efficiently (several key cryptography schemes depend on this fact). The *Strong Church-Turing thesis* expresses the belief that all efficiently computable functions can be efficiently computed by a Turing machine. Quantum computation directly challenges the Strong Church-Turing thesis as fast algorithms have been found for problems, like the factoring problem, that have resisted all attempts at efficient solutions on a Turing machine. Because of the difficulty of proving lower bounds on running time it is not yet known whether the class of all efficiently solvable problems on a quantum computer is larger than its Turing machine counterpart – although this is widely believed to be the case.

There are a couple of significant differences between classical and quantum com-

puters that make the latter seem capable of more computing power. Whereas in a classical computer a representative state is an n-bit string of zeroes and ones, the corresponding general n-qubit state in a quantum computer is represented by a vector in a 2^n-dimensional vector space. The quantum mechanical phenomenon of having a system of particles be in a superposition of several states at once allows a quantum computer to perform an exponential number of computations in a single step. The results, however, remain in a superimposed state and clever quantum computing algorithms find ways to disentangle this state to extract the information sought.

We present Shor's celebrated factoring algorithm and work out the details of the computation as well as the running time. The algorithm is significant in that if implemented it could be used to break several encryption algorithms which depend on the infeasibility of factoring large numbers. We also present an algorithm by Grover which exploits an inherent geometry in quantum states to perform an unstructured search in square root of the time expected. Because of the ubiquitous presence of unstructured search in common classical algorithms, Grover's algorithm holds much potential to be used to improve the running times of existing algorithms.

Superpositions in quantum computing are an endless source of surprising results. We present the No-Cloning and No-Distinguishability theorems which limit the information we may have about our states. We also present the teleportation and superdense coding circuits which, on the other hand, exhibit the great power of quantum computers to transmit information reliably and compactly. We provide a brief history of complexity theory and how it applies to quantum computing, as well as some thoughts on the Strong Church-Turing thesis and its influence on the direction of research.

The last section of the paper consists of research done over the summer of 2004 at an REU at SUNY Potsdam. This work was done in conjunction with three other undergraduate students and under the supervision of Professor Kazem Mahdavi, faculty in mathematics at SUNY Potsdam. The research presented aims to find generators of an abelian group through the efficient quantum computing solution of Simon's Hidden Subgroup problem. Results for both the finite abelian group and the torsion-free abelian group are presented.

1.2 Linear Algebra

To define all the necessary terms used would be cumbersome and unnecessary, so we instead refer the reader to one of the numerous linear algebra texts out there, such as Lang's "Introduction to Linear Algebra" [Lan94]. Instead we focus on introducing just the main concepts used throughout quantum computing. We consider complex vector spaces of the form $V = \mathbb{C}^n$, which are n-tuples of complex numbers. As is done throughout all the literature on quantum computing, we adopt Dirac notation to represent vectors in V, as well as their outer and inner products.

If v is the label for a vector in V, then we write $|v\rangle$ to denote the $n \times 1$ column vector referred to by v and $\langle v|$ to denote $|v\rangle^\dagger$; that is, if $|v\rangle = (z_1, \ldots, z_n)$ is an $n \times 1$ column vector then $\langle v|$ is the $1 \times n$ row vector $|v\rangle^\dagger = [z_1^*, \ldots, z_n^*]$, where z_i^* denotes the complex conjugate of z_i. Note that v itself is not the row or column vector, it is simply a label. For example, $|v\rangle$, $|1\rangle$ and $|0\rangle$ are all valid vectors, as long as the context makes it clear what the labels v, 1, and 0 stand for.

Having Dirac notation and thinking of vectors in V as $n \times 1$ matrices makes the inner product very natural. Let $|v\rangle = (z_1, \ldots, z_n)$ and $|w\rangle = (y_1, \ldots, y_n)$. We write $\langle v||w\rangle$, or simply $\langle v|w\rangle$, to mean the matrix multiplication of $\langle v|$ and $|w\rangle$; that is $\langle v|w\rangle = \sum z_i^* y_i$. We define this to be the *inner product* of $|v\rangle$ and $|w\rangle$. We define the *norm* of a vector labeled by v to be $\sqrt{\langle v|v\rangle}$. We say that a vector is a *unit vector* if its norm is 1, and that the two vectors v and w are orthogonal, or equivalently, perpendicular, if $\langle v|w\rangle = 0$. A set of unit vectors $\{v_i\}$ is said to be *orthonormal* if each pair $\{v_i, v_j\}$ for $i \neq j$ is orthogonal. An orthonormal set $\{v_i\}$ of n elements in V is said to be an *orthonormal basis* for V, and every element $v \in V$ can be written

$$|v\rangle = \sum \langle v_i|v\rangle |v_i\rangle \tag{1.1}$$

in Dirac notation. For the vector space $V = \mathbb{C}^n$ we will use the set $|0\rangle$, $|1\rangle$, ..., $|n-1\rangle$ to denote the standard orthonormal basis of V.

The other concept we need is that of linear operators. A *linear operator* on V is defined as a function $A : V \to V$ such that $A(\sum z_i v_i) = \sum z_i A v_i$ where $z_i \in \mathbb{C}$ and $v_i \in V$. It is easy to see that if $\{v_i\}$ is an orthonormal basis of V, then the values Av_j uniquely define A on all of V. As we saw above, since $\{v_i\}$ is an orthonormal basis, $Av_j = \sum \langle v_i|Av_j\rangle v_i$. If we let $A_{ij} = \langle v_i|Av_j\rangle$ then we may think of A as acting

on elements of V by way of matrix multiplication with composition of operators corresponding naturally to products of matrices. This is a standard idea in linear algebra, and we use it here often. We also define the *outer product* of two vectors $|v\rangle$ and $|w\rangle$ to be the operator $|v\rangle\langle w|$ which acts on another vector, say $|y\rangle$, naturally: $(|v\rangle\langle w|)|y\rangle = \langle w|y\rangle|v\rangle$. It is useful to note that if $|i\rangle$ and $|j\rangle$ are basis vectors in V with i and j labels in $\{0, 1, \ldots, \dim V - 1\}$, then $|i\rangle\langle j|$ represents an operator whose matrix representation consists of all 0's except for a 1 in the i^{th} row and j^{th} column.

Given an operator A define $A^\dagger$ such that $\langle v|Aw\rangle = \langle A^\dagger v|w\rangle$. We call $A^\dagger$ the *adjoint* of A, and in terms of matrix representation $A^\dagger_{ij} = A^*_{ji}$. Note that $(A^\dagger)^\dagger = A$ and so $\langle Av|w\rangle = \langle v|A^\dagger w\rangle$. Of importance in quantum computing are *unitary operators*, which are defined as operators A such that $A^\dagger A = I$, where I is the identity operator. In particular this implies that $\langle Av|Av\rangle = \langle v|A^\dagger Av\rangle = \langle v|v\rangle$, so unitary operators preserve the norm. We also define A to be a *projection operator* if $A = A^\dagger = A^2$. The importance of these two types of operators stems from the postulates of quantum physics and will be explained in the next section.

Without going into much detail it will suffice here to say that given $W = \mathbb{C}^n$ and $V = \mathbb{C}^m$ there is a natural mapping, denoted by $\otimes$, $W \times V \to \mathbb{C}^{mn}$ such that, for $w, w_1, w_2 \in W$, $v, v_1, v_2 \in V$ and $z \in \mathbb{C}$ the following four properties are satisfied:

I $z(w \otimes v) = (zw) \otimes v = w \otimes (zv)$

II $(w_1 + w_2) \otimes v = w_1 \otimes v + w_2 \otimes v$

III $w \otimes (v_1 + v_2) = w \otimes v_1 + w \otimes v_2$

IV $\langle w_1 \otimes v_1 | w_2 \otimes v_2\rangle = \langle w_1|w_2\rangle\langle v_1|v_2\rangle$

We call the new vector space formed in such a way from W and V the tensor product of W and V and denote it $W \otimes V$. It is clear from the above rules that if $\{v_i\}$ is a basis for V and $\{w_j\}$ is a basis for W, then $\{v_i \otimes w_j\}$ is a basis for $V \otimes W$. Lastly, given A and B operators on V and W respectively, we define the operator $A \otimes B$ to be $(A \otimes B)(v \otimes w) = Av \otimes Bw$.

1.3 Quantum Computing Postulates

Quantum mechanics is the theory of the behavior and interaction of atomic and subatomic particles, hereafter referred to as quantum particles. Quantum computing seeks to utilize quantum particles and their peculiar interactions according to the rules laid out by quantum mechanics to perform computational work. Therefore before we get into the details of quantum computing we first introduce four postulates of quantum mechanics that will serve as the foundation for our formal quantum computer. The formulation of quantum computing as presented here is known as the *circuit model* of quantum computing. The statements of the following postulates are taken from Nielsen and Chuang [NC00, p. 80-97] and a more thorough discussion of them may be found there.

Postulate 1. *Associated to any isolated physical system is a complex vector space with an inner product known as the state space of the system. The system is completely described by its state vector, which is a unit vector in the system's state space.*

It must be noted that this postulate refers to an isolated physical system, which in the strictest sense may only be applied to the whole universe. However, practically significant degrees of isolation within a system may be achieved so that this distinction ceases to be of concern. The simplest such system, and also the one taken to be the fundamental building block for quantum computing, is when $V = \mathbb{C}^2$. Such a system has two basis vectors, usually denoted $|0\rangle$ and $|1\rangle$, and is called a qubit. The previous postulate places a restriction on how we may describe a physical system, but it does not say that any system so described is achievable. In fact, there are real physical systems which may be described by qubits, a common example is the spin property of an electron, and it is this attainability in addition to the overall simplicity that make a qubit a good starting point for a quantum computer.

From a simple glance the qubit resembles the classical computer bit which represents the two states 0 or 1, a weak or strong electrical signal. On the other hand as stated in the postulate, any unit vector $|\phi\rangle$ of V is a state, and therefore any $|\phi\rangle = a|0\rangle + b|1\rangle$ with $a, b \in \mathbb{C}$ such that $|a|^2 + |b|^2 = 1$ is a valid state for a qubit to be in. We call such a state a *superposition* of the basis states $|0\rangle$ and $|1\rangle$, and more generally we call a state $|\phi\rangle = \sum a_i|\phi_i\rangle$ the *superposition of the states* $|\phi_i\rangle$ *with amplitudes* a_i. The fundamental ability of quantum particles to be in a superimposed

state is what accounts for the radically different behavior we get when compared to classical computer operations. As we will see later, superposition accounts for both a drastic speed up in operations, and an increased difficulty in extracting valuable information.

Postulate 2. *The evolution of a closed quantum system is described by a unitary transformation. That is, the state $|\phi_0\rangle$ of the system at time t_0 is related to the state $|\phi_1\rangle$ of the system at time t_1 by a unitary operator U which depends only on the times t_0 and t_1,*

$$|\phi_1\rangle = U|\phi_0\rangle \tag{1.2}$$

Once again the system described must be a closed one, yet once again it has been found that closed quantum systems may be achieved to good approximation. Quantum mechanics dictates that in such a closed system any change that occurred could be described by the application of a unitary operator. The primary reasons the operator is chosen to be unitary are to preserve norms and to take valid states to valid states. In addition, a unitary operator is linear, and therefore satisfies the *superposition principle*, which states that the evolution of a superposition must be the superposition of the individual evolutions. Postulate 2, which deals with discrete evolutions in time, can in fact be derived from a more inclusive postulate which deals with a continuous time evolution of a quantum system. For a brief introduction and discussion of this see Nielsen and Chuang [NC00, p. 82]. Because in practice exact precision is not possible, we may not assume that any given unitary operator is easily and accurately constructible. The viability of constructing a given unitary operator and examples of common quantum computing unitary operators will be discussed shortly.

Before presenting the next postulate we first introduce what are called measurement operators. *Measurement operators* are a collection of operators $\{M_m\}$ indexed by some indexing set m such that

$$\sum_m M_m^\dagger M_m = I. \tag{1.3}$$

Postulate 3. *Quantum measurements are described by a collection $\{M_m\}$ of measurement operators acting on the state space of the system being measured. The index m refers to the measurement outcomes that may occur in the experiment. If the state*

of the quantum system is $|\phi\rangle$ immediately before the measurement then the probability that result m occurs is given by

$$p(m) = \langle\phi|M_m^\dagger M_m|\phi\rangle, \tag{1.4}$$

and the state of the system after the measurement is

$$\frac{M_m|\phi\rangle}{\sqrt{\langle\phi|M_m^\dagger M_m|\phi\rangle}}. \tag{1.5}$$

The evolution of a closed quantum system in time is described by postulate 2, but once a researcher decides to measure a given system it is no longer closed. Postulate 3 deals with this issue by formalizing the operations that one is allowed to perform to measure a quantum system. It also introduces another fundamental difference between quantum mechanics and classical physics, that of uncertainty.

In subsequent discussions of measurements, we will be further restricting M_m to be projection operators satisfying

$$M_m M_n = \delta_{m,n} M_m, \tag{1.6}$$

where $\delta_{m,n}$ is the Kronecker delta. It turns out that taken together with the rest of the postulates of quantum mechanics, the extra restriction of measurements to be projective operators is just as powerful as the more general case described in postulate 3 [NC00, p. 87]. It is projection operators that we use most often as they make calculations of the probabilities particularly simple. Since a projection operator satisfies $M = M^\dagger = M^2$, the probability of outcome m is simply

$$p(m) = \langle\phi|M|\phi\rangle. \tag{1.7}$$

Note also, that from equation 1.3 we get that even for the most general measurement operator, the sum of the probabilities of all outcomes is $\sum_m p(m) = \sum_m \langle\phi|M_m^\dagger M_m|\phi\rangle = \langle\phi|I|\phi\rangle = 1$, as one would expect.

From now on we'll use projection operators as our measurement tools, we follow suit with notation and refer to them as P_n. A commonly used example of applying a measurement operator is that of measuring a qubit in its computational basis. Let $P_1 = |0\rangle\langle 0|$ and $P_2 = |1\rangle\langle 1|$ and apply it to the general qubit state of $|\phi\rangle = a|0\rangle + b|1\rangle$ with $|a|^2 + |b|^2 = 1$ as usual. Accordingly, the probability of outcome 1 occurring

is $\langle\phi|P_1|\phi\rangle = |a|^2$ and of outcome 2 occurring is $\langle\phi|P_2|\phi\rangle = |b|^2$. The resulting new states would be $\frac{P_1|\phi\rangle}{|a|} = \frac{a}{|a|}|0\rangle$ and $\frac{P_2|\phi\rangle}{|b|} = \frac{b}{|b|}|1\rangle$ respectively.

The resulting phenomena brought about because of superpositions and projective measurements have some surprising consequences which will be discussed in a later section. Results such as the no clone theorem which states that an arbitrary qubit may not be copied, and the no distinguishability theorem which states that two arbitrary states may not be determined to be different with 100% certainty are cornerstones in the limitations of quantum computing and stem from the restrictive definition of its measurement operators.

Postulate 4. *The state space of a composite system is the tensor product of the state spaces of the component physical systems. Moreover, if we have systems numbered* 1 *through* n*, and system* i *is in state* $|\phi_i\rangle$*, then the joint state of the total system is* $|\phi_1\rangle \otimes |\phi_2\rangle \otimes \cdots \otimes |\phi_n\rangle$*, oftentimes abbreviated* $|\phi_1\rangle|\phi_2\rangle \cdots |\phi_n\rangle$*.*

The physical system representing one qubit is a rather simple one, and it is natural to want to combine more than one of these types of systems to construct a larger computational machine. This is the postulate that allows us to do exactly that. There are a variety of ways we may denote elements of a composite system. Oftentimes it is convenient to consider the computational basis of $V = \mathbb{C}^{2^n}$ and denote it by $|0\rangle, |1\rangle, \ldots, |2^n - 1\rangle$. Sometimes it will be effective to write out every number in binary, with the ith digit of the binary representation comprising the state of the ith composite system. For example, a joint system of 2 qubits with one qubit in the $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$ state and the other in the $\frac{|0\rangle-|1\rangle}{\sqrt{2}}$ state can be represented together as a 2-qubit in the state $\frac{|0\rangle+|1\rangle}{\sqrt{2}} \otimes \frac{|0\rangle-|1\rangle}{\sqrt{2}} = \frac{|00\rangle-|01\rangle+|10\rangle-|11\rangle}{2}$, or likewise, $\frac{|0\rangle-|1\rangle+|2\rangle-|3\rangle}{2}$.

A fundamentally new kind of state can be created through the use of postulate 4: one that is not a tensor product of any two smaller systems of the composite system yet is still a valid state. A simple example of this is in a 2 qubit system, in which $|\phi\rangle = \frac{|00\rangle+|11\rangle}{\sqrt{2}}$ is such a state. It is certainly convincing, and also a simple exercise to check by just assuming the opposite and seeing what happens, that $|\phi\rangle$ cannot be written as a tensor product of two qubits. This type of state is called an *entangled state*, and its peculiarities, as we will see, make it incredibly useful in quantum computation and quantum information.

1.4 Quantum Operators

Unitary operators acting on a state of n-qubits are called *quantum gates*. A finite collection of quantum gates and measurement operators, all acting on the same n-qubit system, is defined to be a *quantum circuit*. A *quantum computer* is then defined as a quantum circuit used for the purpose of computation. Just as in classical computing where we build arbitrarily complicated logical gates from a few simple gates acting on a small number of bits (such as the NOT, OR, and AND gates), in a quantum computer we want to consider only quantum gates which can be built from simpler quantum gates acting on only a few qubits at a time. Note that even though quantum gates act on any valid qubit, which can be a superposition of several basis states, to define them it is sufficient to define their operation on only the basis states as the rest would follow from linearity. Thus, often when defining quantum gates we speak only of their action on the basis states.

We begin by considering the simplest of all quantum gates, those which act on a single qubit. Of these, the most important are the Pauli matrices:

$$X := \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad Y := \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \quad Z := \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, \tag{1.8}$$

and the Hadamard (H), phase (S), and $\pi/8$ (T) gates:

$$H := \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \quad S := \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix} \quad T := \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix}. \tag{1.9}$$

Remember that $a|i\rangle\langle j|$ is the operator whose matrix has the term a in the i^{th} row and j^{th} column, so in outer product notation the Hadamard transform can be written as

$$H = \frac{1}{\sqrt{2}} \sum_{x,y\in\{0,1\}} (-1)^{x\cdot y} |x\rangle\langle y|. \tag{1.10}$$

It is useful to note the effect of the Hadamard transform on $|0\rangle$ and $|1\rangle$, namely:

$$H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \qquad H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle). \tag{1.11}$$

The ability to perform controlled operations, that is operations of the type 'if this occurs, then do that,' is useful in both classical and quantum computing. The simplest such gate is the controlled NOT, or CNOT, gate. As you may know, the

NOT gate takes $|0\rangle$ to $|1\rangle$ and vice versa (the X Pauli matrix accomplishes this). Accordingly, the CNOT gate operates on 2 qubits and applies the NOT gate to the second qubit if the first one is $|1\rangle$. This can be represented by saying that the CNOT gate takes $|x\rangle|y\rangle$ to $|x\rangle|x \oplus y\rangle$, where $\oplus$ denotes addition mod 2. Using the CNOT and Hadamard gates we may construct the *Bell states* for a 2-qubit system. Denoting the CNOT gate by C, the Bell states are:

$$C(H \otimes I)|00\rangle = C\frac{1}{\sqrt{2}}\big(|0\rangle + |1\rangle\big) \otimes |0\rangle = C\frac{1}{\sqrt{2}}\big(|00\rangle + |10\rangle\big) = \frac{1}{\sqrt{2}}\big(|00\rangle + |11\rangle\big) = |\beta_{00}\rangle; \tag{1.12}$$

$$C(H \otimes I)|01\rangle = C\frac{1}{\sqrt{2}}\big(|0\rangle + |1\rangle\big) \otimes |1\rangle = C\frac{1}{\sqrt{2}}\big(|01\rangle + |11\rangle\big) = \frac{1}{\sqrt{2}}\big(|01\rangle + |10\rangle\big) = |\beta_{01}\rangle; \tag{1.13}$$

$$C(H \otimes I)|10\rangle = C\frac{1}{\sqrt{2}}\big(|0\rangle - |1\rangle\big) \otimes |0\rangle = C\frac{1}{\sqrt{2}}\big(|00\rangle - |10\rangle\big) = \frac{1}{\sqrt{2}}\big(|00\rangle - |11\rangle\big) = |\beta_{10}\rangle; \tag{1.14}$$

$$C(H \otimes I)|11\rangle = C\frac{1}{\sqrt{2}}\big(|0\rangle - |1\rangle\big) \otimes |1\rangle = C\frac{1}{\sqrt{2}}\big(|01\rangle - |11\rangle\big) = \frac{1}{\sqrt{2}}\big(|01\rangle - |10\rangle\big) = |\beta_{11}\rangle. \tag{1.15}$$

Bell states are also called *EPR states* or *pairs* after Einstein, Podolsky, and Rosen who worked out many of their strange properties. They are entangled; that is, they are not a tensor product of qubits of a smaller dimension. Clearly, Bell states form an orthonormal basis. As we will see later, they pop up in interesting places to produce surprising results.

In classical computing, logic gates play the role of quantum gates. Practically one desires to construct logic gates of arbitrary complexity. The way to do this is not to devise an electronic circuit from scratch that will produce the necessary outputs given the necessary inputs, but rather to produce a few simple electronic circuits that simulate simple logic gates with few inputs and outputs, and then to construct more complex logic gates out of the simpler ones. There are a few simple logic gates such as NAND, OR, AND and NOT, which are easily constructed in the laboratory and from which any logic gate of arbitrary complexity may be built out of. What we desire is a similar result dealing with quantum gates.

It turns out that there is a similar result for quantum computing. It says that to construct an arbitrary gate on n qubits requires only CNOT gates and gates acting

on only 1 qubit at a time. At first this may seem to be an analogous result to the classical case, however one must note that in the classical case we build arbitrary gates starting with only a finite number of simpler ones whereas here we've required access to an arbitrary 1-qubit gate, the number of which is not finite. If one allows a small error ϵ of measuring a different result than you would if you applied the gate exactly, then an arbitrary gate may be simulated using only the CNOT, Hadamard, phase, and $\pi/8$ gates. For clarity, we restate the two results, which are both proved in Nielsen and Chuang [NC00, p. 188].

Universal Quantum Gates. *An arbitrary gate on n qubits may be constructed out of CNOT and 1-qubit gates.*

Approximate 2-qubit Gates. *An arbitrary 2-qubit gate may be approximated to ϵ precision by CNOT, Hadamard, phase, and $\pi/4$ gates. Measuring the output of the approximated gate guarantees that the probability of each event differs from the desired probability by at most ϵ.*

As a final note it must be said that practically we are only concerned with gates which we can approximate in a polynomial in the input size number of simpler gates. This requires quantum algorithms to verify that the proposed gates can be implemented efficiently. Nielsen and Chuang [NC00, p. 198] show that this is in fact not possible for most gates.

Given an arbitrary unitary operator U on an n-qubit, it is possible to construct efficiently a controlled version of the operator on k additional qubits. The new operator then acts on an $(n+k)$-qubit and applies U to the last n qubits if the first k qubits are in some specific state, usually all $|1\rangle$. A simple application of this is to construct the controlled X Pauli matrix with 2 control bits. The gate would apply the Pauli matrix to the last qubit if and only if the first 2 qubits are both $|1\rangle$. This gate is called the Toffoli gate. Recall that $X|0\rangle = |1\rangle$ and $X|1\rangle = |0\rangle$ and so the Toffoli gate would take $|a\rangle|b\rangle|c\rangle \to |a\rangle|b\rangle|c \oplus ab\rangle$, where $\oplus$ denotes addition mod 2, as usual. Notice that ab is 1 only if both a and b are 1, so c is only changed from 1 to 0 or from 0 to 1 if the first two qubits were both $|1\rangle$. Momentarily letting T denote the Toffoli gate, $T|a\rangle|b\rangle|0\rangle$ and $T|1\rangle|1\rangle|a\rangle$ simulate the classical AND, and NOT gates, respectively. The AND and NOT gates are classically universal; that is, any arbitrarily complex logic gate may be built up of repeated applications of NOT

and AND gates. Thus one immediately gets the result that quantum computers are computationally at least as powerful as their classical counterparts, as expected.

Another very useful quantum gate that we will encounter is the quantum Fourier transform. The most general definition of the quantum Fourier transform isn't always constructible in a polynomial in n number of gates and will not be discussed here. Two special, yet nevertheless very powerful, cases are defined for the groups $\mathbb{Z}_2^n$ and $\mathbb{Z}_N$, where $\mathbb{Z}_N$ is short for the cyclic group on N elements $\mathbb{Z}/N\mathbb{Z}$. For $\mathbb{Z}_2^n$ we get the Hadamard transform tensored with itself n times:

$$H^{\otimes n} = \left(\frac{1}{\sqrt{2}} \sum_{x,y \in \{0,1\}} (-1)^{x \cdot y} |x\rangle\langle y| \right)^{\otimes n} = \frac{1}{\sqrt{2^n}} \sum_{x,y \in \{0,1,\ldots,2^n-1\}} (-1)^{x \cdot y} |x\rangle\langle y| \tag{1.16}$$

where $x \cdot y$ is defined as the dot product when x and y are expressed in binary notation (i.e. for 2-qubits $1 \cdot 3 = 01 \cdot 11 = 0 \cdot 1 + 1 \cdot 1 = 1$). The case of $\mathbb{Z}_N$ is called the *discrete Fourier transform*, and for $|x\rangle$ in the computational basis $|0\rangle, \ldots, |N-1\rangle$

$$F|x\rangle = \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} e^{2\pi i x y / N} |y\rangle. \tag{1.17}$$

If we set $N = 2^n$ and let $x = x_1 x_2 \ldots x_n$ be the binary representation of x, and $0.x_l \ldots x_m$ denote $x_l/2 + x_{l+1}/2^2 + \cdots + x_m/2^{m-l+1}$, then we can also write the Fourier transform in a very useful product representation:

$$F|x_1, \ldots, x_n\rangle = \frac{\left(|0\rangle + e^{2\pi i 0.x_n}|1\rangle\right)\left(|0\rangle + e^{2\pi i 0.x_{n-1}x_n}|1\rangle\right) \cdots \left(|0\rangle + e^{2\pi i 0.x_1 x_2 \ldots x_n}|1\rangle\right)}{\sqrt{2^n}} \tag{1.18}$$

This can be easily verified with some simple algebra. This form for the discrete Fourier transform is oftentimes useful and may lead to a clearer understanding of its effects. There is also a way to use this decomposition to construct an equivalent quantum circuit for the Fourier transform which employs only $O(n^2)$ gates from our discrete set.

Chapter 2

Grover's Search Algorithm

Designing quantum computing algorithms which perform strictly better than their classical counterparts is a hard problem. When an algorithm becomes available that does just that, it is worthwhile to study its methods for generalizations and ways to apply these methods to other scenarios. Grover's search algorithm takes advantage of the beautiful geometry inherent in the representation of qubit states to achieve a result which seems inconceivable classically.

The Grover search problem is formulated in terms of a black box function f. Let X be a set and $S \subset X$ be a subset, called the solution set. Assume we have a function f on X such that $f(x) = 1$ if and only if $x \in S$. We take X as input and think of the input as containing $N = |X|$ different elements and having $M = |S|$ of them be solutions. Our goal is to find an element of the solution set. At first one might find formulating the problem in terms of a black box function to be awkward. It might seem that the black box function already knows what the solutions are and that there is in fact no need for a search algorithm. What the black box function does however is only provide us with a way to recognize the solution. Knowing the solution is quite another matter. For example, given a large number n one might wish to find its factors. A naive algorithm is to simply test all the numbers from 2 to $\sqrt{n}$ and see if they divide n. In this case, the black box function is easy to compute: given k it would employ the division algorithm to test if k divides n. Classically then, it would take $O(\sqrt{n})$ invocations of the black box function to go through testing n for primality. As we will see however, employing Grover's algorithm we would need to evaluate the black box function only $O(\sqrt[4]{n})$ times.

The factoring example is not practically useful as there exist much faster classical, as well as quantum, algorithms to deal with this problem. However, it illustrates the conceptual point that there are cases when knowing how to calculate the black box function is an easy matter, yet the solution set remains unknown. The techniques used in Grover's algorithm together with the techniques from Shor's factoring algorithm comprise the foundation for modern quantum computing algorithm design. In addition, there are many other areas where applying Grover's algorithm directly inside classical algorithms yields a practical speedup, such as speeding up the findings of solutions to certain NP-complete problems [NC00, p. 263].

We begin with a slight simplification of the general search algorithm, one where we assume we know the size M of the solution set. We also assume that $M < N/2$, since if that were not true, then we could pick a random state and find an element of the solution set at least one out of every two times. We assume the implementability of an Oracle operator, $\mathcal{F}$, which is simply the black box function expressed as a quantum operator. Given that $|x\rangle$ is an n-qubit and $|y\rangle$ is a qubit, $\mathcal{F}|x\rangle|y\rangle = |x\rangle|y \oplus f(x)\rangle$, where $\oplus$ denotes addition mod 2. A useful trick that Grover's algorithm employs is that

$$\mathcal{F}|x\rangle \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) = |x\rangle \left(\frac{|0 \oplus f(x)\rangle - |1 \oplus f(x)\rangle}{\sqrt{2}}\right) = (-1)^{f(x)}|x\rangle \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right). \tag{2.1}$$

This formula simply expresses the fact that the $|x\rangle \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right)$ are eigenvectors of $\mathcal{F}$ with corresponding eigenvalues $(-1)^{f(x)}$.

Before describing the algorithm, we first define the Grover operator G as the composition of four simpler operators. It is helpful to note that in general, the operator $2|x\rangle\langle x| - I$ is a reflection across the unit vector $|x\rangle$. This is easily seen by applying the operator to an arbitrary vector $\alpha|x\rangle + \beta|y\rangle$ where $\langle x|y\rangle = 0$:

$$\big(2|x\rangle\langle x| - I\big)\big(\alpha|x\rangle + \beta|y\rangle\big) = \alpha\big(2|x\rangle\langle x|x\rangle - |x\rangle\big) + \beta\big(2|x\rangle\langle x|y\rangle - |y\rangle\big) = \alpha|x\rangle - \beta|y\rangle.$$

The result is that the perpendicular component, the coefficient of $|y\rangle$, changes sign. With this result in mind we define G as the composition of the following four operators.

1 Apply the oracle $\mathcal{F}$ to the $n + 1$-qubit.

2 Apply the Hadamard transform $H^{\otimes n}$ to the first n qubits.

3 Apply a reflection across the vector $|0\rangle$ to the first n qubits.

4 Apply the Hadamard transform $H^{\otimes n}$ to the first n qubits.

The Hadamard transform is easily constructible in the laboratory, and the reflection across the $|0\rangle$ vector can be constructed with an $O(n)$ number of gates acting on 2 qubits at a time. By our previous remarks in the quantum operators section, this makes the operators comprising the Grover operator all feasible. The only concern is the Oracle operator, but since that is assumed to be based on a black box function we don't concern ourselves with its implementation in the general case.

We start the algorithm with an n-qubit tensored with a 1-qubit, both in the 0 basis state. We first modify the starting state into one which will set up for a more natural application of the algorithm. We apply the Hadamard transform to the first n qubits to get a uniform distribution of states

$$|\phi\rangle = H^{\otimes n}|0\rangle = \frac{1}{\sqrt{N}} \sum_{x,y=0}^{N-1} (-1)^{x \cdot y} |x\rangle\langle y||0\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle. \tag{2.2}$$

We also apply HX to the last qubit to get our desired $\frac{|0\rangle - |1\rangle}{\sqrt{2}}$ state. At this point we can begin to analyze the effect of the Grover operator. Keeping in mind that the Hadamard transform is its own inverse, we have that

$$G = H^{\otimes n}(2|0\rangle\langle 0| - I)H^{\otimes n}\mathcal{F} = (2H^{\otimes n}|0\rangle\langle 0|H^{\otimes n} - H^{\otimes n}IH^{\otimes n})\mathcal{F} = (2|\phi\rangle\langle\phi| - I)\mathcal{F}. \tag{2.3}$$

To see more clearly what the Grover operator does it will be helpful to define unit vectors

$$|\alpha\rangle = \frac{1}{\sqrt{N-M}} \sum_{x|f(x)=0} |x\rangle, \tag{2.4}$$

$$|\beta\rangle = \frac{1}{\sqrt{M}} \sum_{x|f(x)=1} |x\rangle, \tag{2.5}$$

where $f(x)$ is our old black box function. We can now write $|\phi\rangle$ as

$$|\phi\rangle = \sqrt{\frac{N-M}{N}}|\alpha\rangle + \sqrt{\frac{M}{N}}|\beta\rangle, \tag{2.6}$$

and think of $|\phi\rangle$ as being in the space spanned by $|\alpha\rangle$ and $|\beta\rangle$. Now, applying the

oracle $\mathcal{F}$ to $|\phi\rangle\big(|0\rangle - |1\rangle\big)/\sqrt{2}$ we get, by 2.1,

$$\mathcal{F}|\phi\rangle\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) = \mathcal{F}\left(\sqrt{\frac{N-M}{N}}|\alpha\rangle + \sqrt{\frac{M}{N}}|\beta\rangle\right)\frac{|0\rangle - |1\rangle}{\sqrt{2}} \tag{2.7}$$

$$= \left(\sqrt{\frac{N-M}{N}}|\alpha\rangle - \sqrt{\frac{M}{N}}|\beta\rangle\right)\frac{|0\rangle - |1\rangle}{\sqrt{2}}. \tag{2.8}$$

This is simply a reflection across $|\alpha\rangle$ in the plane spanned by $|\alpha\rangle$ and $|\beta\rangle$, and subsequently applying $2|\phi\rangle\langle\phi| - I$ will reflect across the vector $|\phi\rangle$. The end result is that applying G once will first reflect across $|\alpha\rangle$ and then across $|\phi\rangle$, producing a rotation in the plane spanned by $|\alpha\rangle$ and $|\beta\rangle$. If we let $\sqrt{\frac{N-M}{N}} = \cos\theta/2$ and $\sqrt{\frac{M}{N}} = \sin\theta/2$, then we can see that the angle between $|\phi\rangle$ and $|\alpha\rangle$ is $\theta/2$. After the reflection done by $\mathcal{F}$, the angle between $|\phi\rangle$ and $\mathcal{F}|\phi\rangle$ is θ, and so reflecting across $|\phi\rangle$ produces a rotation of θ radians. The conclusion is therefore that

$$G|\phi\rangle = \cos\frac{3\theta}{2}|\alpha\rangle + \sin\frac{3\theta}{2}|\beta\rangle, \tag{2.9}$$

and that after applying the Grover operator k times in a row we get

$$G^k|\phi\rangle = \cos\left(\frac{2k+1}{2}\theta\right)|\alpha\rangle + \sin\left(\frac{2k+1}{2}\theta\right)|\beta\rangle. \tag{2.10}$$

With this result in mind, we can state Grover's search algorithm.

Step 1 Start with the $(n+1)$-qubit state of $|0\rangle^{\otimes n}|0\rangle$.

Step 2 Apply $(H^{\otimes n} \otimes H)(I \otimes X)$ to the initial state, where X is the Pauli matrix mentioned in 1.8.

Step 3 Apply the Grover operator k times to the resulting state, where k will be determined as part of the running time analysis.

Step 4 Measure the final answer in the computational basis and apply the black box function to see whether it is part of the solution set. If not, apply the algorithm again.

The last step of the algorithm involves measuring the first n qubits in the computational basis. This is easily achieved by letting the measurement operators be the collection of projection operators $\{P_i\}$, where $P_i = |i\rangle\langle i|$, for $i = 0, 1, \ldots, 2^n - 1$.

We haven't seen yet why Grover's algorithm is in fact better than plain classical algorithms designed to solve this problem. What one is interested in is the number of times, k, you need to apply the Grover operator, which is essentially an application of the black box function, to achieve a state that you can measure to come up with an element of the solution set. In a classical algorithm it is easy to see that since the set of elements is unordered, we must choose on average $\frac{N}{M}$, or $O\left(\frac{N}{M}\right)$, elements before we pick one from the solution set. As we shall see however, Grover's algorithm requires only $k = O\left(\sqrt{\frac{N}{M}}\right)$ calls to the Oracle. Additionally, the probability of not picking an element of the solution set out of our final state is at most $\sqrt{\frac{M}{N}}$.

As we saw, we have that $G^k|\phi\rangle = \cos\left(\frac{2k+1}{2}\theta\right)|\alpha\rangle + \sin\left(\frac{2k+1}{2}\theta\right)|\beta\rangle$, and so we want to find a k such that

$$\cos\left(\frac{2k+1}{2}\theta\right) = 0. \tag{2.11}$$

Solving this, we get $\frac{2k+1}{2}\theta = \frac{\pi}{2}$, or

$$k = \frac{\pi}{2\theta} - \frac{1}{2} = \frac{\pi}{4\arcsin\sqrt{M/N}} - \frac{1}{2}. \tag{2.12}$$

Practically, we would round k to the nearest integer and apply the Grover operator that many times. To find an upper bound for k in simpler terms, note that $k \leq \frac{\pi}{2\theta}$ and recall that $\frac{\theta}{2} \geq \sin\frac{\theta}{2} = \sqrt{\frac{M}{N}}$. Putting these two inequalities together, we get that

$$k \leq \frac{\pi}{4}\sqrt{\frac{N}{M}}. \tag{2.13}$$

Note that for the appropriate k we have that $\left|\frac{2k+1}{2}\theta - \frac{\pi}{2}\right| \leq \frac{\theta}{2}$, and so the probability of measuring an element which isn't part of the solution set is

$$\left(\cos\frac{2k+1}{2}\theta\right)^2 \leq \left(\sin\frac{\theta}{2}\right)^2 = \frac{M}{N}. \tag{2.14}$$

Keeping in mind that the number of Grover operator applications every time the algorithm is run is $O\left(\sqrt{\frac{N}{M}}\right)$, that $\frac{M}{N} \leq \frac{1}{2}$, and that the probability of not getting a solution is at most $\frac{M}{N}$, we can conclude that the expected number of Grover operator applications is still $O\left(\sqrt{\frac{N}{M}}\right)$.

The novel approach of Grover's algorithm is to take a starting superposition, and, viewed as a vector, rotate it closer and closer towards a vector which is the superposition of only the solutions. Earlier, however, we assumed that M, the number of solutions in our set, was known to us. This is not necessary as there exists a quantum algorithm, known as phase estimation, which can determine M in an expected $O(\sqrt{N})$ Oracle calls. The presentation of such an algorithm would take us too far afield, but we note that no classical algorithm exists which can determine the number of solutions in a similar scenario in faster than $O(N)$ time.

Because of its generality Grover's algorithm may be used to speed up many existing algorithms which depend on finding a verifiable solution out of an unordered database of possible solutions. As a more concrete example recall the naive factoring algorithm we discussed at the beginning of this section. It is simple to check whether k is a factor of a given n, and the possible candidates range from 2 to $\sqrt{n}$. This is a natural set up to apply Grover's algorithm. Whereas classically all we could do is go through every candidate and check for divisibility, taking up $O(\sqrt{n})$ time, since $O\left(\sqrt{\frac{N}{M}}\right) = O\left(\sqrt{N}\right)$ and $N = \sqrt{n}$, using Grover's algorithm we could implement the search in expected $O(\sqrt[4]{n})$ time.

Chapter 3

Shor's Factoring Algorithm

Shor's factoring algorithm demonstrated the potential practical importance of quantum computing. Given N an odd number which isn't the power of a prime, Shor's factoring algorithm is capable of finding a factor of N in time polynomial in $\log N$, which is a significant improvement over the best known classical algorithm today. There are no known classical algorithms which achieve this task in faster than superpolynomial time. Several encryption algorithms depend on the fact that it takes a long time for all known algorithms to factor a large natural number N, and if a computer was built capable of implementing Shor's factoring algorithm, then these encryption schemes would be broken.

Shor's algorithm is based upon the following number theoretic facts, which are stated without proof. For a detailed discussion of these ideas see any Number Theory text, such as Hardy and Wright [HW68, p. 48]. Given an N and y, we define (y, N) to be the greatest common divisor of N and y. Euclid's division algorithm can be used to compute this value in $O(\log^2 N)$ time [Knu97, pp. 333-379]. Therefore if we pick a y such that $(y, N) > 1$ we're done since we've found a factor of N. Because of this we may assume that $(y, N) = 1$. We define $y \pmod N$ to be the positive remainder when y is divided by N. There exists a smallest positive integer r, called the *period* of y, such that $y^r = 1 \pmod N$. Now if $r = 2s$, we can write $y^r = 1 \pmod N$ as $(y^s - 1)(y^s + 1) = 0 \pmod N$. Clearly $y^s - 1 \neq 0 \pmod N$ because otherwise s would be the period. If $y^s + 1 \neq 0 \pmod N$, then either $(y^2 + 1, N) > 1$, or $(y^s - 1, N) > 1$ and we may use Euclid's division algorithm to find the common factor.

For reference, we restate the necessary variables and conditions used in Shor's

algorithm.

- We begin with an odd natural number N which is not the power of a prime
- We randomly pick a $y < N$ such that $(y, N) = 1$
- We let r be the period of $y \pmod N$
- For Shor's algorithm to succeed we require the following two necessary conditions

$$r = 2s; \tag{3.1}$$

$$y^s + 1 \neq 0 \mod N. \tag{3.2}$$

Shor's algorithm proceeds by first choosing a y relatively prime to N, or else we're done, and attempting to calculate its period r. Classically calculating r is hard and essentially requires raising y to powers and checking to see if it's 1 (mod N). However on a quantum computer it is possible to apply an exponentiation operator to a superpositioned state and effectively calculate y^k for all k in one step. The task then becomes to manipulate this result to attempt to magnify the amplitudes of the states which can tell us something about r. Once we've determined what r is, if conditions 3.1 and 3.2 are satisfied then a factor of N can be calculated efficiently through the use of Euclid's division algorithm. An analysis of the algorithm shows that the output yields the correct value of r with a high probability, and that the above conditions are also satisfied with a high probability so that an overall reasonable running time is expected.

We first present the steps of Shor's algorithm without much explanation, and then follow them through with more detail and an analysis of running time. We are given an odd N, which we suspect can be factored, and we would like to find one of these factors. We proceed as follows:

Step 1 Choose an n such that $N^2 \leq S = 2^n < 2N^2$, and pick an arbitrary $y < N$. If $(y, N) > 1$, then we're done, so we assume $(y, N) = 1$.

Step 2 Start with 2 n-qubits tensored together, both in their 0 basis state.

Step 3 Apply $H^{\otimes n}$ to the first n-qubit, and then unitarily (to be explained shortly) evaluate the function

$$f(k) = y^k \pmod{N}.$$

Step 4 Apply the discrete Fourier transform (see 1.17) to the first n-qubit.

Step 5 Measure the first n-qubit in its computational basis and attempt to extract r from the result.

Step 6 If r cannot be determined, then return to *Step 1.* If r is odd, then return to *Step 1.* If $r = 2s$ is even but $y^s = -1 \pmod{N}$, return to *Step 1.* Otherwise, use the Euclidean algorithm to compute $(y^s - 1, N)$ and $(y^s + 1, N)$ and if either is bigger than 1, quit. Otherwise, go back to *Step 1* and repeat.

Our first consideration is the feasibility of implementing the given operators. The discrete Fourier transform is implementable in $O(n^2)$ quantum gates acting on only 2 qubits at a time. For an implementation, see almost any book on quantum computing, for example [Pit00]. In *Step 3* we apply the function $f(k) = y^k \pmod{N}$ unitarily. What we mean is that we apply the unitary operator U_f to two n-qubits which takes a basis state $|x\rangle|y\rangle$ to $|x\rangle|y \oplus f(x)\rangle$,

$$U_f|x\rangle|y\rangle = |x\rangle|y \oplus f(x)\rangle, \tag{3.3}$$

where $\oplus$ denotes addition mod 2^n. In our particular case, we apply the operator to a state $|x\rangle|0\rangle$ and therefore get as our result $|x\rangle|f(x)\rangle$. As is standard in most quantum algorithms, we first apply the Hadamard transform to the first qubit to create a uniform superposition of all the basis states. After this is done, applying the unitary implementation of $f(k)$ applies the function to every basis state in the superposition. As for the feasibility of implementing the function $y^k \pmod{N}$, see Vedral et al. [VBE96]

After choosing an appropriate n, start with the tensor product of two n-qubit states, $|\phi_0\rangle = |0\rangle|0\rangle$. *Step 3* calls for the application of the Hadamard operator on the first n-qubit, followed by an application of the unitary operator implementing $f(k)$. We obtain

$$|\phi_1\rangle = U_f(H^{\oplus n} \otimes I)|\phi_0\rangle = U_f \frac{1}{\sqrt{S}} \sum_{k=0}^{S-1} |k\rangle|0\rangle = \frac{1}{\sqrt{S}} \sum_{k=0}^{S-1} |k\rangle|f(k)\rangle. \tag{3.4}$$

So now that we've applied $f(k)$ to every state, we attempt to exploit f's periodicity to magnify the amplitudes of states which can tell us something about r. The operator which does this is the discrete Fourier transform, mentioned in *Step 4*. Remember that the discrete Fourier transform acts by taking a basis state $|k\rangle$ to $\frac{1}{\sqrt{S}}\sum_{u=0}^{S-1} e^{2\pi iku/S}|u\rangle$. Applying this to $|\phi_1\rangle$ we get

$$|\phi_2\rangle = (F \otimes I)|\phi_1\rangle = (F \otimes I)\frac{1}{\sqrt{S}}\sum_{k=0}^{S-1}|k\rangle|f(k)\rangle = \frac{1}{S}\sum_{k=0}^{S-1}\sum_{u=0}^{S-1} e^{2\pi iku/S}|u\rangle|f(k)\rangle \quad (3.5)$$

$$= \frac{1}{S}\sum_{u=0}^{S-1}|u\rangle\sum_{k=0}^{S-1} e^{2\pi iku/S}|f(k)\rangle \quad (3.6)$$

Set

$$s_m = \left\lceil \frac{S-m}{r} \right\rceil \quad (3.7)$$

and note that $f(k)$ is periodic with period r, so if we write $k = m + rj$ for $0 \leq m < r$ and $0 \leq j < s_m$, then $f(k) = f(m + rj) = f(m)$. We can therefore rewrite $|\phi_2\rangle$ as

$$|\phi_2\rangle = \frac{1}{S}\sum_{u=0}^{S-1}|u\rangle\sum_{k=0}^{S-1} e^{2\pi iku/S}|f(k)\rangle = \frac{1}{S}\sum_{u=0}^{S-1}|u\rangle\sum_{m=0}^{r-1}\sum_{j=0}^{s_m-1} e^{2\pi i(m+rj)u/S}|f(m+rj)\rangle \quad (3.8)$$

$$= \frac{1}{S}\sum_{u=0}^{S-1}|u\rangle\sum_{m=0}^{r-1} e^{2\pi imu/S}|f(m)\rangle\sum_{j=0}^{s_m-1} e^{2\pi irju/S} \quad (3.9)$$

$$= \sum_{u=0}^{S-1}|u\rangle\sum_{m=0}^{r-1} b_{um}e^{2\pi imu/S}|f(m)\rangle \quad (3.10)$$

where $b_{um} = \frac{1}{S}\sum_{j=0}^{s_m-1} e^{2\pi irju/S}$.

This is the final state of the quantum algorithm, and we measure this state in the computational basis of the first n-qubit; that is, the set of measurement operators that we use is $\{P_i = |i\rangle\langle i| \otimes I\}$. Before we begin the measurement however, we might wonder, even if the algorithm correctly calculates r, how often can we expect to have r be even and $y^s + 1 \neq 0 \pmod N$. Through an elementary but slightly involved procedure it can be shown that picking y randomly and uniformly among values relatively prime to N yields the above two conditions at least $9/16$ of the time, given that N is odd and not a power of a prime [Hir04]. Consequently, the only remaining concern is whether we can determine the value of r given the result of our

measurement. Shor's algorithm works by being able to compute the value of r with probability at least $\frac{\delta}{\log\log N}$ if the state measured, $|u\rangle$, is such that

$$|ur - dS| < \frac{r}{2} \tag{3.11}$$

for some integer d. This result is practical as δ is large. In fact $\delta > \frac{1}{4}$ if r is at least 19. This means that we want u to be the nearest integer to $\frac{dS}{r}$, while still being within the bounds of $0 \leq u < S$, which results in about r such u's (in fact $r+1$ if $\frac{S}{r}$ is not an integer, and r if it is).

The next step then is to calculate how likely we are to pick such a u as described above. Before that however, we work out a few simple lemmas that will be used in the calculations ahead.

Lemma 3.0.1. *With the notation described above*

$$|b_{um}|^2 = \frac{1}{S^2}\frac{\sin^2(\pi u r s_m/S)}{\sin^2(\pi ur/S)} \tag{3.12}$$

where we use limiting values if $\sin\left(\frac{\pi ur}{S}\right) = 0$.

Proof. Recall that $b_{um} = \frac{1}{S}\sum_{j=0}^{s_m-1} e^{2\pi irju/S}$ and so

$$b_{um} = \frac{1}{S}\sum_{j=0}^{s_m-1} e^{2\pi irju/S} = \frac{1}{S}\frac{e^{2\pi irus_m/S}-1}{e^{2\pi ru/S}-1}, \tag{3.13}$$

using limiting values if $e^{2\pi ur/S} = 1$. Also note that $|b_{um}|^2 = b_{um}b_{um}^*$, and so

$$|b_{um}|^2 = \frac{1}{S^2}\left(\frac{e^{2\pi irus_m/S}-1}{e^{2\pi ru/S}-1}\right)\left(\frac{e^{2\pi irus_m/S}-1}{e^{2\pi ru/S}-1}\right)^* = \frac{1}{S^2}\frac{2-(e^{2\pi iurs_m/S}+e^{-2\pi iurs_m/S})}{2-(e^{2\pi iur/S}+e^{-2\pi iur/S})} \tag{3.14}$$

$$= \frac{1}{S^2}\frac{1-\cos(2\pi urs_m/S)}{1-\cos(2\pi ur/S)} \tag{3.15}$$

$$= \frac{1}{S^2}\frac{\sin^2(\pi rus_m/S)}{\sin^2(\pi ur/S)}. \qquad \square$$

Lemma 3.0.2. *The function* $g(y) = \frac{\sin y}{y}$ *is decreasing on the interval* $[0,\pi]$. *In particular, if* $|y| \leq \delta$ *then* $g^2(y) \geq g^2(\delta)$.

Proof. It will suffice to show that $g'(y) < 0$ in the appropriate domain. The statement about $g^2(y)$ is easily seen from the fact that $g^2(-y) = g^2(y)$. We have

$$g'(y) = \frac{y\cos y - \sin y}{y^2} \tag{3.16}$$

It is clear that this derivative is negative for the range $[\frac{\pi}{2}, \pi]$, so it remains to show that $y \cos y - \sin y < 0$ for $y \in \left(0, \frac{\pi}{2}\right)$. In this range both $\cos y$ and $\sin y$ are positive, so it is equivalent to show that $y < \tan y$. These two functions are equal at 0, and once again taking a derivative we can reduce it to showing that $1 < 1 + \tan^2 y$, which is obvious. □

Lemma 3.0.3. *With the notation described above assume $|ur - dS| \leq \frac{r}{2}$ for some integer d, and set $t = \frac{ur}{S} - d$. Then*

$$-\frac{\pi}{2}\left(1 + \frac{1}{N-2}\right) < \pi s_m t < \frac{\pi}{2}\left(1 + \frac{1}{N-2}\right) \tag{3.17}$$

Proof. Using the assumption, multiplying by s_m and dividing by S, we obtain

$$-\frac{s_m r}{2S} \leq \frac{u r s_m}{S} - d s_m = t s_m \leq \frac{s_m r}{2S}. \tag{3.18}$$

Recall that s_m was defined as $s_m = \left\lceil \frac{S-m}{r} \right\rceil$, and since $m < r$, we must have

$$s_m - 1 < \frac{S-m}{r} \leq \frac{S}{r} \leq \left\lceil \frac{S-m}{r} \right\rceil + 1 = s_m + 1. \tag{3.19}$$

Inverting 3.19 and multiplying by s_m, we obtain

$$\frac{s_m}{s_m + 1} \leq \frac{s_m r}{S} < \frac{s_m}{s_m - 1} = 1 + \frac{1}{s_m - 1}. \tag{3.20}$$

From *Step 1* of Shor's algorithm we know that $N^2 \leq S$, and since $r < N$, we conclude from 3.19 that

$$N - 2 < \frac{S}{r} - 2 \leq s_m - 1. \tag{3.21}$$

Putting 3.18, the right-hand side 3.20 and 3.21 together and multiplying by π, we get

$$-\frac{\pi}{2}\left(1 + \frac{1}{N-2}\right) < \pi s_m t < \frac{\pi}{2}\left(1 + \frac{1}{N-2}\right). \tag{3.22}$$

□

Lemma 3.0.4. *For $x > 0$*

$$\sin^2 \frac{\pi}{2}(1+x) > 1 - \left(\frac{\pi}{2} x\right)^2. \tag{3.23}$$

Proof. At 0 both sides are 1. It then suffices to show that the derivative of the difference is always positive. This derivative is

$$\pi \sin\left(\frac{\pi}{2}(1+x)\right) \cos\left(\frac{\pi}{2}(1+x)\right) + \frac{\pi^2}{2} x = \frac{\pi}{2} \sin\big(\pi(1+x)\big) + \frac{\pi^2}{2} x \tag{3.24}$$

$$= -\frac{\pi}{2}\big(\sin(\pi x) - \pi x\big) > 0 \tag{3.25}$$

since $\sin y < y$ for all positive y. □

Lemma 3.0.5. *Given a procedure which succeeds with probability $P(success) = \frac{1}{f(N)}$, where $f(N) > 1$, repeating the procedure $f(N)$ times guarantees achieving success at least once with a probability at least $P^{f(N)}(success) \geq 1 - e^{-1} \approx .63$.*

Proof. The probability that the procedure fails for every one of the $f(N)$ repetitions is $\left(1 - \frac{1}{f(N)}\right)^{f(N)}$, and so the probability that it succeeds at least once must be

$$P^{f(N)}(\text{success}) = 1 - \left(1 - \frac{1}{f(N)}\right)^{f(N)}. \tag{3.26}$$

By elementary calculus we know that $\left(1 - \frac{1}{M}\right)^M$ is an increasing sequence and approaches e^{-1} as M goes to infinity, so

$$\left(1 - \frac{1}{M}\right)^M \leq e^{-1}. \tag{3.27}$$

Combining equations 3.26 and 3.27 we get that

$$P^{f(N)}(\text{success}) \geq 1 - e^{-1}. \tag{3.28}$$

□

We now return to the measurement of our final state. We want to consider the probability $P(u)$ of measuring a state $|u\rangle$ such that $|ur - dS| \leq \frac{r}{2}$. Recall that $|\phi_2\rangle = \sum_{u=0}^{S-1} |u\rangle \sum_{m=0}^{r-1} b_{um} e^{2\pi i m u/S} |f(m)\rangle$, and that the probability of measuring a state $|u\rangle$ using the standard basis projection $P_u = |u\rangle\langle u|$ is

$$P(u) = \langle P_u\phi_2 | P_u\phi_2\rangle = \sum_{m=0}^{r-1} |b_{um}|^2 = \frac{1}{S^2} \sum_{m=0}^{r-1} \frac{\sin^2\left(\frac{\pi u r s_m}{S}\right)}{\sin^2\left(\frac{\pi u r}{S}\right)} \tag{3.29}$$

by lemma 3.0.1. By periodicity we know $\sin^2(x + y\pi) = \sin^2 x$ for any integer y, and so the above can be rewritten as

$$P(u) = \frac{1}{S^2} \sum_{m=0}^{r-1} \frac{\sin^2\left(\pi(ur - dS)s_m/S\right)}{\sin^2\left(\pi(ur - dS)/S\right)} = \frac{1}{S^2} \sum_{m=0}^{r-1} \frac{\sin^2\left(\pi t s_m\right)}{\sin^2\left(\pi t\right)} \tag{3.30}$$

where $t = \frac{ur - dS}{S}$. Because $\sin y \leq y$ for $y \geq 0$, we can conclude that

$$\frac{1}{\sin^2(\pi t)} \geq \frac{1}{(\pi t)^2}. \tag{3.31}$$

Likewise, for $g(y) = \frac{\sin y}{y}$, we obtain

$$\sin^2\left(\pi t s_m\right) = \left(\pi t s_m\right)^2 g^2(\pi t s_m) \geq \left(\pi t s_m\right)^2 g^2\left(\frac{\pi}{2}\left(1 + \frac{1}{N-2}\right)\right) \tag{3.32}$$

$$= \left(\pi t s_m\right)^2 \frac{4}{\pi^2}\left(\frac{N-2}{N-1}\right)^2 \sin^2\left(\frac{\pi}{2}\left(1 + \frac{1}{N-2}\right)\right) \tag{3.33}$$

by lemmas 3.0.3 and 3.0.2. Then, by lemma 3.0.4

$$\sin^2(\pi t s_m) \geq (\pi t s_m)^2 \frac{4}{\pi^2}\left(\frac{N-2}{N-1}\right)^2\left(1-\frac{\pi^2}{4}\frac{1}{(N-2)^2}\right). \tag{3.34}$$

Putting all of this together then, by 3.30, 3.31 and 3.34 the probability $P(u)$ of measuring state $|u\rangle$, given that $|ur - dS| \leq \frac{r}{2}$ is

$$P(u) \geq \frac{1}{S^2}\sum_{m=0}^{r-1}\frac{1}{(\pi t)^2}(\pi t s_m)^2 \frac{4}{\pi^2}\left(\frac{N-2}{N-1}\right)^2\left(1-\frac{\pi^2}{4}\frac{1}{(N-2)^2}\right). \tag{3.35}$$

So, since $s_M \geq \frac{S}{r} - 1$ and $\frac{4}{\pi^2} \geq .4$ we get

$$P(u) \geq \frac{.4}{r} \tag{3.36}$$

for N bigger than about 1000. This is the probability of picking one particular u satisfying 3.11, but as we noted in the paragraph following 3.11, there are about r such u's. We may conclude then that the total probability of picking an arbitrary u satisfying 3.11 is at least .4.

So once we've measured an appropriate u, we must extrapolate a value for r. It turns out this isn't always possible even if all the above conditions are met, but it happens often enough that the running time of the algorithm is still satisfactory. If we measure a state $|u\rangle$ such that $|ur - dS| \leq \frac{r}{2}$, then, rewriting this and keeping in mind that $N^2 \leq S$, we have that $|\frac{u}{S} - \frac{d}{r}| \leq \frac{1}{2S} \leq \frac{1}{2N^2} \leq \frac{1}{2r^2}$. This says that the fraction $\frac{d}{r}$ approximates the known fraction $\frac{u}{S}$ to within $\frac{1}{2r^2}$. However, there can be at most one such approximation of $\frac{u}{S}$, and it can be found in time polynomial in $\log N$ using a continued fraction expansion of $\frac{u}{S}$. For a proof of the previous claim and details on continued fractions, see for example Hardy and Wright [HW68, p. 129].

The correct fraction approximating $\frac{u}{S}$ can be found through a continued fraction expansion, and then the denominator can be extracted. However, this denominator will only be r if $\frac{d}{r}$ is in lowest terms. The problem then comes down to seeing how often d is relatively prime to r. This can be summarized by asking for a lower bound of $\frac{\phi(r)}{r}$, where ϕ is Euler's totient function and $\phi(n)$ equals the number of positive numbers less than r and relatively prime to r. It turns out that $\frac{\phi(r)}{r} \geq \frac{\delta}{\log\log r} \geq \frac{\delta}{\log\log N}$, for some constant δ (where δ is at least $\frac{1}{4}$ if we restrict $r > 19$). The reader is once again referred to Hardy and Wright [HW68, p. 267] for the details. What this result implies is that by lemma 3.0.5 for an appropriate y, we need only repeat the

procedure $\log\log N$ times to get a high probability of determining the correct r. Since we can pick an appropriate y at least $\frac{9}{16}$ of the time [Hir04], and the procedure yields an appropriate u with a probability of at least .4, we need only run the algorithm $O(\log\log N)$ times.

As a wrap up, we summarize the previous discussion.

1. Given an N to factor, choose $y < N$.

2. Compute $d = (y, N)$ using Euclid's division algorithm. If $d > 1$, return d as a factor.

3. Compute the order r of y using Shor's factoring algorithm.

4. If the computed value of r is correct, even, and $y^{\frac{r}{2}}+1 \neq 0 \pmod{N}$ then proceed to compute $d_1 = (y^{\frac{r}{2}}+1, N)$ and $d_2 = (y^{\frac{r}{2}}-1, N)$ and output whichever is bigger than 1.

Step 2 can be done $O(\log^2 N)$ time. *Step 3* involves implementing the Hadamard transform, which takes $O(\log N)$ time, the function y^k, which takes $O(\log^3 N)$ time [VBE96, p. 5], the Fourier transform which takes $O(\log^2 N)$ time, and computing the continued fraction convergents, which takes $O(\log^2 N)$ time. *Step 4* once again takes only $O(\log^2 N)$ time. Overall the algorithm is guaranteed a success probability of at least $O\left(\frac{1}{\log\log N}\right)$ and so by lemma 3.0.5 repeating the whole procedure $\log\log N$ times produces an algorithm which takes $O\left(\log^3 N \log\log N\right)$ time and yields a factor of N with probability at least $1 - e^{-1}$.

Chapter 4

On the Power and Limitations of Quantum Computing

In addition to algorithms which are capable of significantly faster computing times on classical problems, quantum computers are also capable of achieving results which wouldn't even seem plausible classically. However, because of the peculiar nature of superpositions of states along with the increased flexibility there come fundamental limitations as well.

4.1 Distinguishability

The first result we present is known as the No-Distinguishability Theorem. It helps to phrase the setup in terms of an information game between two players, A and B, Alice and Bob. Suppose that there is a set of states $\{|\phi_i\rangle\}$ for $1 \leq i \leq n$ known to both players. Alice picks one of these states, $|\phi_j\rangle$, and Bob's task is to determine the value of j reliably, that is with probability 1. In other words, Bob must find which state was picked. Classically this problem is one we never even bother to consider. All Bob would really have to do is test the state he got from Alice for equality against the known set of all states by looking at the component bits. In quantum computing however, the equivalent to looking at the component bits of a state is to look at the magnitudes of basis states making up any given state. This however is an impossibility, as measuring a state doesn't illuminate much about the magnitudes of the component basis states.

Nevertheless, one might think that even given this limitation there might exist a set of measurement operators which would allow Bob to identify Alice's state. It turns out that as long as the given set of states that Alice can pick from is mutually orthogonal, then Bob can identify every state. All he needs to do in this case is simply apply the measurement operators $P_i = |\phi_i\rangle\langle\phi_i|$ and $P_0 = I - \sum P_i$ to Alice's state since $\langle\phi_i|P_j\phi_i\rangle = \delta_{ij}$ where δ_{ij} is the Kronecker delta. This isn't surprising since if all the $|\phi_i\rangle$ are mutually orthogonal then they can be thought of as a subset of an orthonormal basis, and the states of an orthonormal basis are in a one to one correspondence with the simple bit states of a classical computer. As soon as the game ventures into quantum territory however, all hope of distinguishability is lost. It turns out that having just a pair of nonorthogonal states is enough to ensure that Bob cannot win. We now state and prove this result more formally [Gud03, p. 191].

No-Distinguishability Theorem. *Given a state $|\phi_i\rangle$ taken from a set of two nonorthogonal states $|\phi_1\rangle$ and $|\phi_2\rangle$ there can be no measurement which will determine i to be* 1 *or* 2 *with probability* 1*.*

Proof. Suppose that such a measurement exists, and let it be $\{P_i : 1 \leq i \leq n\}$. If the outcome of a measurement is j, then it must be possible to say with certainty whether state $|\phi_1\rangle$ or $|\phi_2\rangle$ was measured. Therefore there must exist a function $f(j)$ such that $f(j) = 1$ if the outcome j implies that $|\phi_1\rangle$ was measured, and $f(j) = 2$ if the outcome j implies that $|\phi_2\rangle$ was measured. Define

$$Q_i = \sum_{j|f(j)=i} P_j \tag{4.1}$$

for $i = 1, 2$. Note that by 1.6 we have

$$Q_i = Q_i^\dagger = Q_i^2. \tag{4.2}$$

Since the measurement is assumed to be able to tell one state from the other with probability 1, we must have that

$$\langle\phi_1|Q_1\phi_1\rangle = \langle\phi_2|Q_2\phi_2\rangle = 1. \tag{4.3}$$

However, by the definition of a quantum measurement, we must have that $Q_1 + Q_2 = I$ and so

$$\langle\phi_1|Q_1\phi_1\rangle + \langle\phi_1|Q_2\phi_1\rangle = \langle\phi_1|I\phi_1\rangle = 1. \tag{4.4}$$

By 4.2 and 4.4 we get that

$$\langle Q_2\phi_1|Q_2\phi_1\rangle = \langle \phi_1|Q_2^\dagger Q_2\phi_1\rangle = \langle \phi_1|Q_2\phi_1\rangle = 0. \tag{4.5}$$

Hence $|Q_2\phi_1\rangle = 0$. Write $|\phi_2\rangle = a|\phi_1\rangle + b|\psi\rangle$ where $\langle\psi|\psi\rangle = 1$, $\langle\psi|\phi_1\rangle = 0$, and $|a|^2+|b|^2 = 1$. Then $|b| < 1$ since $a = \langle\phi_1|\phi_2\rangle \neq 0$. On the other hand $Q_2|\phi_2\rangle = bQ_2|\psi\rangle$ and

$$\langle\psi|Q_2\psi\rangle \leq \langle\psi|Q_2\psi\rangle + \langle\psi|Q_1\psi\rangle = 1. \tag{4.6}$$

Hence

$$\langle\phi_2|Q_2\phi_2\rangle = \langle Q_2\phi_2|Q_2\phi_2\rangle = |b|^2\langle\psi|Q_2\psi\rangle \leq |b|^2 < 1. \tag{4.7}$$

This is in direct contradiction with 4.3. Therefore no such measurement exists. □

Though simple, this is an important and fundamental result. No measurement exists which can reliably distinguish two nonorthogonal states from each other. On the other hand, as we saw, it is quite simple to construct a measurement which distinguishes orthogonal states from each other. What this implies is that in terms of distinguishability, a quantum computer is as powerful as a classical one, but not more so.

4.2 Cloning

The No-Distinguishability Theorem is closely related to another fundamental limitation theorem of quantum computing, the No-Cloning theorem. In classical computing, given a bit $|i\rangle$ where i is either 0 or 1, it is easy to make a copy of i by simply applying the CNOT gate to $|i\rangle|0\rangle$. If i were 0, then the second bit wouldn't get flipped and would remain 0, whereas if i were 1, then the second bit would get flipped and become 1 as well. The result is then that we take as input $|i\rangle|0\rangle$ and get as output $|i\rangle|i\rangle$. It's illuminating to consider what happens when one tries to apply the same procedure to copy an arbitrary quantum qubit state. If we let $|\phi\rangle = a|0\rangle + b|1\rangle$, then applying the CNOT gate to $|\phi\rangle|0\rangle$ we get $a|0\rangle|0\rangle + b|1\rangle|1\rangle$, which is almost always *not* equal to $|\phi\rangle|\phi\rangle = a^2|0\rangle|0\rangle + ab|0\rangle|1\rangle + ab|1\rangle|0\rangle + b^2|1\rangle|1\rangle$. In fact, with the No-Distinguishability Theorem in mind, one gets the suspicion that it should be fundamentally impossible to clone an arbitrary state. If this were not so, then given an arbitrary state $|\phi_i\rangle$ out of two known nonorthogonal states $|\phi_1\rangle$ and $|\phi_2\rangle$, we would

be able to first clone $|\phi_i\rangle$ as many times as we wanted and then estimate $|a|$ and $|b|$, where $|\phi_i\rangle = a|0\rangle + b|1\rangle$, to arbitrary precision by simply measuring the cloned states in the computational basis. Through this procedure we could then determine which state $|\phi_i\rangle$ was to arbitrary precision. This isn't the same as saying that we can distinguish the states with probability 1, but it is sufficiently close to make one suspect that such a result is not possible. This suspicion is warranted, and we now state the formal result.

No-Cloning Theorem. *It is impossible to clone an unknown quantum state* $|\phi\rangle$.

Proof. Assume that we begin with a quantum machine that has two slots, an input and an output slot. The input slot takes the unknown state $|\phi\rangle$ and the output slot starts in some standard state $|s\rangle$ and attempts to reproduce $|\phi\rangle$ without modifying the original input. More formally, we assume we have a unitary operator U such that $U|\phi\rangle|s\rangle = |\phi\rangle|\phi\rangle$. Suppose that this unitary operator works for at least two different input states, $|\phi\rangle$ and $|\psi\rangle$, so

$$U|\phi\rangle|s\rangle = |\phi\rangle|\phi\rangle \tag{4.8}$$

$$U|\psi\rangle|s\rangle = |\psi\rangle|\psi\rangle. \tag{4.9}$$

Taking the inner product of 4.8 with 4.9 we get

$$\big(\langle\phi|\psi\rangle\big)^2 = \langle s|s\rangle\langle U\phi|U\psi\rangle = \langle U\phi|U\psi\rangle = \langle\phi|\psi\rangle \tag{4.10}$$

which implies that $\langle\phi|\psi\rangle = 0$ or 1, or in other words, either $|\phi\rangle = |\psi\rangle$ or $|\phi\rangle \perp |\psi\rangle$. □

It is important to pay attention to exactly what this result says. It is specific in noting that an *arbitrary* state cannot be cloned. It is conceivable that a state out of a certain collection of orthogonal states may be cloned. In particular, the proof of the theorem suggests that a cloning operator which clones $|\phi\rangle$ has no fundamental objection to cloning any other state which is orthogonal to $|\phi\rangle$. This is fortunate since we'd like to be able to reproduce as many copies of some orthonormal basis as we want, and this theorem does not pose a threat to our ability to do so. In fact, in the algorithms so far presented, we usually assume that we start with the state $|0\rangle^{\otimes n}$ and then proceed to carry out the algorithm and measure the result. If we don't get a satisfactory solution, we repeat the algorithm. What remains unsaid however is our ability to repeatedly start with the state $|0\rangle^{\otimes n}$. It is in fact rather

simple to design quantum circuits to copy any state in an orthonormal basis (CNOT gate equivalent for the basis as described above), as is expected, since an orthonormal basis is equivalent to classical bits.

4.3 Quantum Teleportation

Just as superposition presents fundamental limitations on our ability to distinguish and produce arbitrary states, it also provides us with surprising new techniques of information processing. The first result presented is known as quantum teleportation. It is a lofty title, but it is well deserved. Through the use of an entangled EPR pair or Bell state it is possible to transmit classical information which would let Bob reproduce an arbitrary qubit of Alice. We proceed to discuss this procedure more fully.

Suppose Alice and Bob meet and together produce $|\beta_{00}\rangle = \frac{|00\rangle+|11\rangle}{\sqrt{2}}$ as defined in 1.12. Alice takes the first qubit and Bob takes the second, and then they go their separate ways. At some subsequent time Alice receives an unknown qubit $|\phi\rangle$ that she wants to transmit to Bob. She doesn't, however, have a quantum information channel to simply send the qubit and can only send Bob classical information, bits. Is this even possible? As we've seen by the no cloning theorem, Alice has no way of determining what $|\phi\rangle$ is as she's restricted to only one copy of it. Also, not only does Alice not know the state of $|\phi\rangle$ but even if she did, $|\phi\rangle$ is described in $\mathbb{C}^2$, and she would only be able to transmit the data to a fixed precision. The surprising answer however is that yes, this is in fact possible due to the peculiar behavior of quantum states, and in particular EPR states.

We examine what occurs when one measures an EPR state like $|\beta_{00}\rangle = \frac{|00\rangle+|11\rangle}{\sqrt{2}}$. Because all 2-qubit states are physically made up of two qubits, the qubits can be physically separated. Yet the state these two particular qubits are in is mathematically inseparable into the product of the individual states. Recall that when the joint state of several qubits cannot be expressed as the tensor product of their individual states, the qubits are said to be in an entangled state.

Assume that Alice has the first qubit and that Bob has the second qubit. Since β_{00} is an entangled state the fate of Alice's qubit is tied to that of Bob's. Consider what happens when Alice measures her qubit in the computational basis. Alice applies the

measurement made up of the projections $P_0 = |0\rangle\langle 0|$ and $P_1 = |1\rangle\langle 1|$. Since her qubit is part of an entangled state however, Alice essentially applies the measurements $P_0 = |0\rangle\langle 0| \otimes I$ and $P_1 = |1\rangle\langle 1| \otimes I$. From Postulate 3 we can conclude that the only possible states of the entangled state after the measurement is applied are $|00\rangle$ or $|11\rangle$. Notice then that if Alice measures a $|0\rangle$ then Bob's qubit also degenerates into $|0\rangle$, and likewise for $|1\rangle$. So if Bob were to measure his qubit after Alice had measured hers, he would always get the same answer that Alice had gotten, whereas if he made the measurement before she did he would have a $\frac{1}{2}$ probability of measuring $|0\rangle$ and $\frac{1}{2}$ probability of measuring $|1\rangle$. It is exactly this property of entangled states which makes them useful, as well as rather mysterious. As we now exhibit, quantum teleportation exploits this useful feature of EPR states to accomplish the task of sending Bob Alice's unknown state.

The Procedure We start off with Alice taking the first qubit and Bob taking the second qubit of the state

$$|\beta_{00}\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}} \tag{4.11}$$

Then Alice receives an unknown qubit

$$|\phi\rangle = a|0\rangle + b|1\rangle \tag{4.12}$$

that she wants to transmit. The system is then defined by a 3-qubit state of which the first qubit is the unknown qubit $|\phi\rangle$ that Alice received, the second qubit is part of the Bell state and is owned by Alice, and the third qubit is also part of the Bell state and is owned by Bob. Our initial state is therefore

$$|\psi_0\rangle = |\phi\rangle \otimes |\beta_{00}\rangle = a|0\rangle \left(\frac{|00\rangle + |11\rangle}{\sqrt{2}}\right) + b|1\rangle \left(\frac{|00\rangle + |11\rangle}{\sqrt{2}}\right) \tag{4.13}$$

The next step is for Alice to send her two qubits through a CNOT gate.

$$|\psi_1\rangle = (CNOT \otimes I)|\psi_0\rangle = \frac{1}{\sqrt{2}}\Big[a|0\rangle\big(|00\rangle + |11\rangle\big) + b|1\rangle\big(|10\rangle + |01\rangle\big)\Big] \tag{4.14}$$

The next step involves applying the Hadamard (see 1.11) transform to the first qubit.

$$|\psi_2\rangle = (H \otimes I \otimes I)|\psi_1\rangle = \frac{1}{2}\Big[a\big(|0\rangle + |1\rangle\big)\big(|00\rangle + |11\rangle\big) + b\big(|0\rangle - |1\rangle\big)\big(|10\rangle + |01\rangle\big)\Big] \tag{4.15}$$

$$= \frac{1}{2}\Big[|00\rangle\big(a|0\rangle + b|1\rangle\big) + |01\rangle\big(a|1\rangle + b|0\rangle\big) + |10\rangle\big(a|0\rangle - b|1\rangle\big) + |11\rangle\big(a|1\rangle - b|0\rangle\big)\Big] \tag{4.16}$$

At this point Alice proceeds to measure her two qubits in the computational basis. Recall that after measurement the quantum state under observation collapses to what was observed. Given that the result of Alice's measurement is on the left, the following represents the state that Alice and Bob's 3-qubit system collapses to:

$$00 \rightarrow |00\rangle\big(a|0\rangle + b|1\rangle\big) \tag{4.17}$$

$$01 \rightarrow |00\rangle\big(a|1\rangle + b|0\rangle\big) \tag{4.18}$$

$$10 \rightarrow |00\rangle\big(a|0\rangle - b|1\rangle\big) \tag{4.19}$$

$$11 \rightarrow |00\rangle\big(a|1\rangle - b|0\rangle\big) \tag{4.20}$$

At this point notice that after the measurement the information from Alice's unknown state has moved to Bob's state. As soon as Alice transmits the results of her measurement to Bob, Bob can use the above equations to manipulate his state into the original $|\phi\rangle$. The manipulation is simple and uses the X and Z Pauli matrices. Recall that X flips the qubit from $|0\rangle$ to $|1\rangle$ and vice versa, and Z changes the sign of the $|1\rangle$ qubit. It is then easy to check that if Alice's measurement results are xy then Bob needs to apply the operators $Z^x X^y$ to his qubit the recover $|\phi\rangle$.

So what have we achieved? Given a preparatory phase where Alice and Bob get to share two qubits of an entangled state, we have achieved the transfer of a qubit of quantum information through the transfer of two bits of classical information. A common question that arises once one is introduced to quantum teleportation is whether quantum teleportation allows transmission of quantum states, or information, faster than the speed of light. After all, given that Bob and Alice are sufficiently far away, once Alice applies her set of operations Bob's state has already gained the essential qualities of Alice's unknown state. This is not quite so however, because though it is true that Bob's state changes accordingly when Alice makes a measurement, Bob gains no information until Alice transfers to him the two bits of classical information. This transfer is the bottleneck of the operation and makes sure that the procedure does not in fact transfer information faster than the speed of light. Also, quantum teleportation is not in violation of the No-Cloning Theorem because it does not make a copy of $|\phi\rangle$ as the original is destroyed in the process.

Quantum teleportation has been a very useful tool in quantum computing. It has shown that a shared EPR state and two classical bits is a resource containing

at least as much information as one qubit. In this sense it exhibits the ability of quantum computing to employ different resources in carrying information around. Many other methods of resource exchange have been developed based on quantum teleportation. In addition, quantum teleportation has proved useful in both error correcting codes as well as the construction of quantum gates which are *a priori* resistant to noise [NC00, Ch. 10,12].

4.4 Superdense Coding

While quantum teleportation demonstrated the ability to carry a qubit of information through a shared EPR state and two bits, superdense coding demonstrates how to carry 2 bits of classical information through the use of 1 qubit (and easily generalizes to $2n$ bits through n qubits). Alice would like to send Bob 2 bits of classical information and achieves this by first manipulating and then sending him her 1 qubit of information, which he then measures for the result.

The procedure follows a similar scheme to that of quantum teleportation but in reverse. We start with Alice and Bob sharing the two qubits that make up

$$|\phi\rangle = |\beta_{00}\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}}. \tag{4.21}$$

Arbitrarily, say that Alice and Bob have agreed on the same numbering scheme for the Bell states that we have presented. The idea is for Alice to manipulate her qubit locally so as to produce one of the four Bell states, then to send her qubit over to Bob who would measure the 2-qubit state to determine which Bell state was produced and thus determine the 2 bits of information that Alice sent. Since Bell states are mutually orthogonal this measurement is possible. Below are the specifications for which operations to apply depending on which information Alice wants to send. Refer back to 1.8 and 1.12 for definitions of the Pauli matrices and Bell states.

$$00 \rightarrow (I \otimes I)|\phi\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}} = |\beta_{00}\rangle \tag{4.22}$$

$$01 \rightarrow (X \otimes I)|\phi\rangle = \frac{|10\rangle + |01\rangle}{\sqrt{2}} = |\beta_{01}\rangle \tag{4.23}$$

$$10 \rightarrow (Z \otimes I)|\phi\rangle = \frac{|00\rangle - |11\rangle}{\sqrt{2}} = |\beta_{10}\rangle \tag{4.24}$$

$$11 \rightarrow (iY \otimes I)|\phi\rangle = \frac{|01\rangle - |10\rangle}{\sqrt{2}} = |\beta_{11}\rangle. \tag{4.25}$$

As we noted, this same technique can be used to send $2n$ bits of classical information by transmitting n qubits. All Alice and Bob have to do is share n EPR pairs all tensored together. Through exactly the same operations Alice can operate on each EPR pair to produce one of the four Bell basis states, together producing one of 4^n basis states, which translates to a $2n$ bit of information. This result is fascinating but by no means unique as there are many more like it in quantum information theory. For more detail and a good list of further reading, see Nielsen and Chuang [NC00, Ch. 12].

The previous results on distinguishability, cloning, teleportation, and superdense coding give a good preliminary overview of some of the fundamental properties that arise from the superposition of quantum states. As we saw, the No-Cloning Theorem restricts us to starting with only known orthonormal basis states. Superdense coding exhibits the utility of EPR pairs in transmitting information while working under the restrictions of the No-Distinguishability Theorem. Quantum teleportation also employs EPR pairs, but this time in order to pass quantum information through classical channels with minimal preparation beforehand.

Chapter 5

Complexity Theory

5.1 A Selected History

Before complexity theory there was the theory of computation. In 1936 Turing introduced his model for theoretical computation, what is now known as a Turing machine, which has stood the test of time as the commonly accepted definition of what a computable function is. Turing's model was a machine with access to infinite sequential memory, a finite alphabet of data, and a finite number of possible states. The machine would function by having a head read off and manipulate symbols on the tape, one entry at a time. Though seemingly limited at first, the Turing machine has captured everything that we have so far considered computable, and has been shown to be at least as powerful as every reasonable model of computation ever suggested. It has become the de facto definition of computable.

It took some time, however, before complexity theory would evolve from Turing machines. Complexity theory owes its beginnings to a 1965 paper by Hartmanis and Stearns, "On the Computational Complexity of Algorithms" [HS65]. This paper introduced the notions of time and space complexity based on the size of the input. It also used these notions to show that, roughly, given more time and space, more can be computed. Before the paper by Hartmanis and Stearns there had been minimal work done on complexity, but once the general definition for time and space complexity became available a slew of work ensued. Many results over the next few years dealt with exhibiting hierarchies of time and space complexity. For example Hennie and Stearns showed in a 1966 paper titled "Two-tape simulation of multitape Turing ma-

chines" [HS99] that if $t_1(n) \log t_1(n) = o\big(t_2(n)\big)$, then there are problems which could be done in $O\big(t_2(n)\big)$ time but not in $O\big(t_1(n)\big)$ time. Many variations on the Turing machine, such as multitape and nondeterministic Turing machines, were examined. Though all of these variations had already been shown to be equivalent in their power of computation to the standard Turing machine, much work was now being done in examining the related time and space complexities of the different models.

Almost immediately after Hartmanis and Stearns' paper researchers began developing formulations of efficient computation. It was noted that all reasonable deterministic models of computation could be reduced to the standard Turing machine in polynomial time. In a 1965 paper entitled "Maximum matchings and a polyhedron with 0,1-vertices" [Edm65a] and a subsequent one entitled "Paths, trees and flowers" [Edm65b] Edmonds noted the wide array of problems solvable in polynomial time and provided an informal discussion of those solved in non-deterministic polynomial time. His papers began what evolved into a guiding force in complexity theory, the **P** vs **NP** problem.

One of the most elegant ways of formulating many complexity classes is through what are called *decision problems*. Decision problems are simply problems with only a yes or no answer. Many of these are stated through the use of a *formal language*. A formal language L is a subset of all finite strings formed given some finite alphabet Σ. Usually, we take $\Sigma = \{0, 1\}$, and so an example of a language L might be the language of all primes $L = \{10, 11, 101, 111, \ldots\}$. Many common and interesting problems can be rephrased as decision problems. For example, factoring is captured in the following:

Factoring. *Given two integers m and n with $m < n$, does n have a non-trivial factor less than m?*

Given the decision problem formulation, it is easy to define **P** and **NP**. If we associate a language L with its decision problem, then problems in **P**, which stands for polynomial, are simply those for which there exists a Turing machine which can recognize whether a string x is in L or not in time polynomial in the length of x. Problems in **NP** then, are those for which if x is in L then there exists a string w, called the *witness*, which can ease the verification that x is in L. A witness string is thought of as a string which is used to quickly verify whether x is a member of L.

Stated a little more formally, the requirements for a language to be in **NP** is that given a string x and a string w there must exist a Turing machine which can recognize whether w proves that x is in L or not, and does so in polynomial time. Note that if w proves that x is in L then x is, in fact, in L. However, if w doesn't prove that x is in L, we don't know whether x is in L or not, and so this witness scheme doesn't actually provide a polynomial time algorithm to solve the decision problem associated with the language.

As an example we can apply the above definition of **NP** to see that the factoring problem is a member of this class. If we are asked if there is a factor of n which is less than m and we are given w as a witness, then all we have to do is check to see that $w < m$ and that $w|n$. Both of these operations can be done in time linear in the input size, and so factoring must be in **NP**. As noted, if we are given a witness w for which either $w \not< m$ or $w \nmid n$, then we still don't know if the factoring problem for m and n is a yes or no, but only that w doesn't work as a witness.

The biggest question in complexity theory right now, and perhaps all of theoretical computer science, is whether $\mathbf{P} = \mathbf{NP}$. That is, are there any problems in **NP** which cannot be solved in polynomial time? The problem looks like it should have a quick and simple answer, but turns out to have deep subtleties that make proofs hard to come by. An important subclass of **NP** are the **NP**-complete problems. **NP**-complete problems are those to which any **NP** problem can be reduced in polynomial time. What this means is that if an **NP**-complete problem is solved in polynomial time, then performing an additional polynomial number of steps one can use this solution to solve any other **NP** problem, also in polynomial time. This essentially makes the **NP**-complete problems the hardest problems of the **NP** class and practically speaking, once a problem is recognized as being **NP**-complete, all attempts at solving it exactly are abandoned and instead partial and approximating solutions are attempted.

It is not obvious that there should even be any **NP**-complete problems, but it turns out there are. In 1971, Cook, in a paper titled "The complexity of theorem-proving procedures," [Coo71] proved that the satisfiability problem (SAT), which asks whether a Boolean expression containing and, not, or and parentheses, is satisfiable (whether there is an assignment of variables to make the expression evaluate to true) is **NP**-complete. Proving that at least one problem was **NP**-complete made proving others much simpler. Instead of showing that any **NP** problem can be reduced to

some given problem one now only needed to show that some **NP**-complete problem can be reduced to the given problem. Similar results soon flooded in and Karp in 1972 in his paper titled "Reducibility among combinatorial problems" [Kar72] proved the **NP**-completeness of 8 central combinatorics problems, such as the traveling salesman, clique, and set cover problems. Karp's paper served as a template for many, now standard, techniques of proving **NP**-completeness, and since then thousands of problems have been proven to be **NP**-complete.

A few natural problems have been found which are not in **NP** and in 1976 Stockmeyer defined the polynomial time hierarchy (**PH**) in his paper "The polynomial-time hierarchy" [Sto76]. The lowest level is defined as **P**, and the second level is **NP**. You get to a subsequent level by requiring problems on that level to have witnesses which are verifiable on the current level. It is believed that the hierarchy is strict and that every level is a proper subset of the one above it; however this hasn't been proved. Another big and central complexity class introduced was **PSPACE**, which is the class of all problems solved in a polynomial amount of space, regardless of the amount of time taken.

It is clear that **P** is in **PSPACE** since if you only take a polynomial number of steps you can only traverse a polynomial amount of space. **NP** is also in **PSPACE**. To see this, note that for any given input size n there must be a polynomial $p(n)$ that bounds the size of the witness. The size of the witness must be polynomial since by definition of the **NP** class the witness must be verified in **P**, and as we noted, $\mathbf{P} \subseteq \mathbf{PSPACE}$. So then for an input of size n we may simply test all $2^{p(n)}$ possible witnesses, making sure we reuse the space after each witness is tested. Since each witness runs in **P**, it must also run in **PSPACE**, so $\mathbf{NP} \subseteq \mathbf{PSPACE}$. This same argument can be used inductively to show that the whole polynomial hierarchy **PH** is in **PSPACE**.

There are many more complexity classes and a plethora of partial results about them. However, not many fundamental questions have been resolved. Two more basic complexity classes are **L**, for logarithmic space with no time constraints, and **EXP**, for exponential time with no space constraints. Together, the classes all satisfy

$$\mathbf{L} \subseteq \mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE} \subseteq \mathbf{EXP}. \tag{5.1}$$

It is known according to space and time hierarchy theorems that **L** is a proper subset

of **PSPACE** and that **P** is a proper subset of **EXP**. This implies that at least one of the inclusions above is strict. Which one, however, is still an open question.

5.2 Quantum Complexity Theory

Compared with the rest of complexity theory quantum complexity theory is still very young. It was not until 1985 that Deutsch even introduced a theoretical framework for quantum computers in the form of a quantum Turing machine (QTM). In that paper Deutsch showed the existence of a universal QTM, one which can take as input both another QTM and an input string, and run the input string through the given QTM. This showed that a QTM model was sufficiently powerful to capture its own description and could be used analogously to the Turing machine.

Deutsch's paper was principally concerned with showing that the QTM was a valid model for computation and did not provide an efficient implementation of a universal QTM. It was not until Bernstein and Vazirani's 1993 paper (updated version [BV97]) that quantum computational complexity became a full fledged topic. Bernstein and Vazirani showed an efficient universal QTM construction which was capable of simulating the work of an arbitrary QTM with only a polynomial time reduction. They also showed that QTM's may be treated as discrete devices in that though theoretically they employ infinite precision amplitudes and transition probabilities, in practice one does not need very precise approximations of these amplitudes and transition probabilities to achieve the desired outputs. Soon after, in a 1993 paper entitled "Quantum Circuit Complexity" [Yao90], Yao showed that the quantum circuit model, which is the model used in this paper, can also be polynomially reduced to the QTM, thereby equating the computational power and complexity of both models.

Bernstein and Vazirani's paper also introduced the **BQP** complexity class of problems. **BQP** consists of all bounded error, quantum, polynomial time algorithms and is the analog of **BPP**, which is the class of all bounded error, polynomial time algorithms on a probabilistic Turing machine. Since polynomial time is widely considered to be the domain of efficiency, **BQP** is the class of *tractable* problems on a QTM; that is, those which can be solved efficiently. *Intractable* problems are those which do not fit into either **BPP** or **BQP** and are said to run in superpolynomial time. Bernstein

and Vazirani exhibited an oracle problem which they proved could be done in polynomial time on a QTM but not in $o(n^{\log n})$ time on a probabilistic Turing machine. This served as evidence that **BQP** is most likely strictly greater than **BPP**; however because this was an oracle result assuming the existence of a black box function, it does not actually prove the above claim in general.

Though it is still an open question whether **BQP** contains **BPP**, there have been three major results that support it. It was Feynman's 1982 paper "Simulating physics with computers" [Fey82] which originally brought to attention the problem of the inefficient, that is superpolynomial, implementation of quantum mechanics simulations that classical computers seemed incapable of improving upon. It is partly with this motivation in mind that Deutsch introduced the theoretical quantum computer. With the universal QTM, efficient quantum mechanical simulations are possible. In an influential paper Shor [Sho97] demonstrated efficient quantum solutions to two other problems which are both widely believed not to be in **BPP**, the factoring and discrete log problems.

In their updated paper Bernstein and Vazirani [BV97, def. 8.1.1] introduced the complexity class **EQP**, for exact, quantum, polynomial time algorithms. In this class, algorithms must always return the correct answer, that is, with probability of error 0. It is clear that $\mathbf{P} \subseteq \mathbf{EQP}$ since quantum computers may always simply operate on the basis states, never venturing into superpositions. However, as is common in complexity theory, it is not known whether the inclusion is strict. Brassard and Høyer [BH97] show that Simon's hidden subgroup problem, which employs a black box function, can be solved exactly on a quantum computer in polynomial time, whereas it is known that the problem can only be solved classically in superpolynomial time. This is evidence for **P** being proper in **EQP**; however it does not prove it because of the oracle involved in Simon's hidden subgroup problem.

5.2.1 Church-Turing Thesis

The Church-Turing thesis started out as an attempt to capture the essence of computable functions by stating that

> **Church-Turing Thesis** Every function which would naturally be regarded as computable can be computed by a Turing machine.

It is clearly not a precise mathematical statement because of the ambiguity of what one would consider to be a naturally computable function and thus cannot be proven. Over the years, however, a substantial quantity of different computational models have been shown to be equivalent to the Turing machine. This has caused the thesis to be almost universally accepted, and many view it as the definition of what computable should mean.

With the extension of the theory of computation to complexity theory, many tried extending the Church-Turing thesis as well. One of the first things considered was the idea of efficient implementation. When complexity theory was just starting out, efficient was taken to mean computable in polynomial time on a deterministic Turing machine, that is, in class **P**. However, in 1977 in a paper entitled "A fast Monte-Carlo test for primality" Solovay and Strassen exhibited a probabilistic algorithm which answered the primality question with a high probability and ran in expected polynomial time. Soon after many other algorithms were exhibited to be in this new class, **BPP**, of algorithms which ran in expected polynomial time. Because no proof exists that $\mathbf{P} = \mathbf{BPP}$, and in fact most believe that $\mathbf{P} \neq \mathbf{BPP}$, the notion of an efficient algorithm was redefined to mean solvable in polynomial time on a probabilistic Turing machine with bounded error. That is, the problem must be in **BPP**. Bernstein and Vazirani [BV97, p. 1] combined this new idea with an already existing extension of the Church-Turing thesis:

> **Strong Church-Turing Thesis** Any reasonable model of computation can be efficiently simulated on a probabilistic Turing machine.

Bernstein and Vazirani take reasonable to mean physically realizable. Though this statement isn't as widely accepted as the bare Church-Turing thesis nor was it discussed by either Church or Turing, it has served as a guiding point for complexity theory research. It is an important statement which incorporates the concerns of both theoretical and practical computer science. On the theoretical side the thesis states that one model of computation, the probabilistic Turing machine, is enough to capture the notion of efficient computation. This implies that the researcher need only worry about probabilistic Turing machines when proving the tractability or intractability of problems. On the practical side, by requiring reasonable to mean physically realizable, we exclude esoteric models which might have theoretical advantages but no

practical ones. This is an important exclusion. For example, analog computers were thought to be theoretically superior to digital ones, yet their non-ideal behavior in the real world eliminated any advantage they held theoretically.

Just as probabilistic Turing machines posed a challenge to the notion of efficient computation and moved it to the realm of **BPP**, quantum computers pose a challenge to this new notion, threatening to move it to **BQP**. Unlike for analog computers, it has been shown for quantum computers that in practice if noise can be kept below a certain threshhold then error correcting codes may be used to efficiently bring the noise level down even more. Physical realizations of quantum computers have been fast improving and it is a promising prospect that the theory can be implemented.

Shor's factoring and discrete log algorithms are both in **NP**. If it can be shown that $\mathbf{P} \neq \mathbf{NP}$, which is what most researchers believe, then it can be said that quantum computers are more efficient than deterministic Turing machines. However, as we've said, **BPP** is now considered to be the real class of efficiently computable functions, and this would not resolve that issue. Fortunately, no known algorithms exist that place factoring into **BPP**, and indeed it is suspected that there are none. This accurately reflects the state complexity theory as a whole is in: many things are suspected and strong evidence abounds, yet absolute proofs are hard to come by.

Another indication that proving the superiority of quantum computation would be hard came when Bernstein and Vazirani [BV97] showed that

$$\mathbf{P} \subseteq \mathbf{BPP} \subseteq \mathbf{BQP} \subseteq \mathbf{PSPACE}.$$

Optimistically, one might say that quantum computing can be used to show that $\mathbf{P} \neq \mathbf{PSPACE}$, a major open question. Pessimistically however, one could say that it will be a long time before anyone shows the theoretical superiority of quantum computing as even **P** vs **PSPACE** has proven to be a hard question. In a subsequent paper, Bennet et al. [BBBV97] provided strong evidence that $\mathbf{NP} \not\subseteq \mathbf{BQP}$, which, if true, would leave many interesting problems out of the realm of efficient quantum computation. Note that although **BQP** contains factoring and the discrete log problems, both are strongly believed not to be **NP**-complete.

So what happens if quantum computers do indeed turn out to violate the Strong Church-Turing thesis? Practically speaking, not much. On the theoretical level a vast amount of research would most likely be dedicated to classifying exactly how

big the class **BQP** is. The whole notion of a computational-model independent complexity theory would have to be reexamined, and the question must be asked whether 'updating' the Strong Church-Turing thesis as was done with probabilistic Turing machines would be better than scrapping it altogether.

All the potential of quantum computing comes from the specific algorithms that can be found. Quantum cryptography processes hold some security advantages over classical ones and distributed EPR pairs can provide for quick and safe information transfer. Unfortunately the main interest in Shor's algorithms stems from the negative effects these algorithms would have on the current cryptography situation and one cannot expect much long term practical use out of them. Perhaps the most important application, though, is to quantum mechanics simulations themselves. Classical algorithms have been excruciatingly slow in simulating quantum mechanics whereas quantum computers hold much potential in this area. Efficient circuits for quantum mechanical simulations have been demonstrated, and quantum computers may one day be an indispensable tool of the experimental physicist.

Chapter 6

Finding Generators of Abelian Groups

This chapter presents some of the results found during a Research Summer Experience at SUNY Potsdam over the summer of 2004. The work was done by of undergraduate students under the supervision of Dr. Kazem Mahdavi, a professor of Mathematics at SUNY Potsdam. The focus was on applications of quantum computing in group theory, and we attempted to extract a minimal generating set from an abelian group.

We first present three algorithms for finding generators of a finite abelian group. We make use of a quantum algorithm known as Simon's Hidden Subgroup Problem and some classical results about generating sets that we state without proof. We then proceed to consider the problem of finding generators of a torsion-free abelian group and present some probability results.

6.1 Finite Abelian Group

We denote by $\mathbb{Z}_n$ the cyclic group $\{0, 1, 2, \ldots, n-1\}$ with addition modulo n. We now state more clearly the problem we intend to solve.

Finite Abelian Group Problem. *Let G be a finite abelian group. Assume that we know the decomposition of G into cyclic groups*

$$G = \mathbb{Z}_{m_1} \oplus \mathbb{Z}_{m_2} \oplus \cdots \oplus \mathbb{Z}_{m_n}, \tag{6.1}$$

and possess a function ϕ which, given an element of G, returns that element's decomposition into components based on the group structure. Given a black box addition operator M defined as

$$M|j\rangle|k\rangle = M|j\rangle|j+k\rangle \tag{6.2}$$

for $j, k \in G$, where $+$ denotes addition in G, find a minimal generating set for G.

Clearly, M embodies addition inside G.

Before we proceed we introduce some elementary notation. For any set S, let $|S|$ denote the cardinality of S. For any set $X \subset G$ let $\langle X \rangle$ denote the subgroup of G generated by the elements of X, and for $a \in G$ let $\langle a \rangle = \langle \{a\} \rangle$. Also, for $a \in G$, let $|a|$ denote the order of a in G, and also note that $|a| = |\langle a \rangle|$.

Two of the presented algorithms will attempt to reconstruct the generating set for G by first computing generating sets for each p-subgroup of G. In order to do that we will need the following result.

Proposition 6.1.1. *Let G be a finite abelian group such that*

$$G = A_{p_1} \oplus A_{p_2} \oplus \cdots \oplus A_{p_k}$$

where each A_{p_i} is a p_i-subgroup of G for distinct primes p_i. Assume that we are given a minimal generating set X_i for each A_{p_i}. Then there exists a procedure to construct a minimal generating set for G requiring time linear in $\max(|X_1|, |X_2|, \ldots, |X_k|)$.

Proof. Let X_{ij} denote the jth element of X_i. It is a standard result of group theory that given two elements a and b of an abelian group such that $(|a|, |b|) = 1$, we have $\langle a \rangle \oplus \langle b \rangle \cong \langle a+b \rangle$. Using induction we can then show that $\{X_{11}, X_{21}, \ldots, X_{k1}\}$ generates the same subgroup as does the single element $\sum_{i=1}^{k} X_{i1}$ since the orders of the elements are powers of distinct primes. Similarly, we can add the second elements of each X_i (when they exist) together, and so on. In fact, it is clear that the number of generators required to generate G using this method is the same as for the p-subgroup of G that requires the greatest number of generators, namely $\max(|X_1|, |X_2|, \ldots, |X_k|)$. □

We proceed to present three algorithms to solve the stated problem. The first algorithm to be presented computes the generators of each p-subgroup by employing the quantum solution to Simon's Hidden Subgroup problem and then combines the

generators via the method presented in Proposition 6.1.1. The second algorithm also computes the generators of the p-subgroups but this time through a random sampling of each p-subgroup. The third algorithm is based entirely on unstructured random sampling of G, but produces comparable results to the first two.

6.1.1 Using Simon's Hidden Subgroup Problem

One strategy for finding the generators of a finite abelian group is to employ the algorithm used to solve Simon's Hidden Subgroup problem. Simon's Hidden Subgroup problem when applied to abelian groups solves the following:

Simon's Hidden Subgroup Problem. *Given an abelian group G which is isomorphic to $\mathbb{Z}_{m_1} \oplus \mathbb{Z}_{m_2} \oplus \cdots \oplus \mathbb{Z}_{m_n}$, and a function $f : G \to G$ with $H_0 \subseteq G$ such that f is constant on H_0 and distinct on its cosets, find the generators for H_0.*

This problem is solved in part using two classically available results which are useful to us as well and we state without proof [BH97]. The second of these results is concerned with an orthogonal subgroup to a given subgroup. What this means precisely is not important to us, and we content ourselves in knowing that constructing the orthogonal subgroup is well defined and self-inverting.

Proposition 6.1.2. *There exists a classically deterministic algorithm that, given a subset X of an abelian group G, returns a linearly independent subset of G that generates the subgroup $\langle X \rangle$, provided we are given a decomposition of G into cyclic subgroups. The algorithm runs in time polynomial in n, where n is the number of cyclic groups in G's decomposition, and linear in $|X|$.*

Proposition 6.1.3. *There exists a classical deterministic algorithm that, given a linearly independent subset X of an abelian group G, returns a linearly independent subset of G that generates the orthogonal subgroup of $\langle X \rangle$, provided a decomposition of G into cyclic subgroups is given. The algorithm runs in time polynomial in n, the number of cyclic groups in G's decomposition.*

Simon's Hidden Subgroup Problem is solved through a quantum circuit which employs the Fourier transform and the given function f to achieve, in one invocation of each, a state which is the superposition of all the elements of $H_0^{\perp}$, which is the

orthogonal subgroup to H_0. The algorithm then proceeds to sample from this set of elements of $H_0^\perp$ and employing Proposition 6.1.2 finds a minimal generating set for $H_0^\perp$. Then, applying Proposition 6.1.3 we end up with a minimal generating set for $(H_0^\perp)^\perp = H_0$.

Our algorithm proceeds according to the following steps, which we explain more thoroughly afterwards.

Step 1 Construct a function which is constant on a given p-subgroup A_p of G and distinct on its cosets.

Step 2 Apply the methods in Simon's Algorithm to find a superimposed state of the elements in $A_p^\perp$.

Step 3 Measure the superimposed state enough times to guarantee with desired probability that we have a generating set for $A_p^\perp$.

Step 4 Apply the algorithms described in Propositions 6.1.2 and 6.1.3 to find a generating set for the p-subgroup A_p.

Step 5 Repeat the above process for all p-subgroups to find the generating sets for each. Then use the procedure described in Proposition 6.1.1 to find a minimal generating set for G.

Beginning with Step 1, we'd like to construct a function which is constant on A_p and distinct on its cosets. The first step is to identify the identity element $|e\rangle$ of G. This is done simply by applying the black box addition operator M to some initial state $|j\rangle|j\rangle$ enough times. Once we've identified the identity element we let $f|j\rangle = M^{|A_p|}|j\rangle|e\rangle = |j\rangle|j^{|A_p|}\rangle$. Since A_p is the subgroup of elements whose orders are a power of p, clearly $|A_p|$ is also a power of p. So the kernel of f is exactly A_p and f satisfies the appropriate conditions.

At this point we apply the procedure in Simon's Hidden Subgroup Problem to find a superposition of all the elements in $A_p^\perp$. For details of this procedure see Brassard and Høyer's paper [BH97]. According to Step 3 we would like to measure our state. A natural question is then how many times might we expect to measure this state before arriving at a generating set for $A_p^\perp$. Given a set of elements X such that $\langle X\rangle$ is a proper subset of $A_p^\perp$, we must have that $\frac{|\langle X\rangle|}{|A_p^\perp|} \leq \frac{1}{2}$ and so we expect to sample twice

before increasing our generating set. If $w \notin \langle X \rangle$ then $|\langle w, X \rangle| \geq 2|\langle X \rangle|$, and so we only have to increase our generating set at most $\log |A_p^{\perp}| \leq \log |G|$ times. We apply Proposition 6.1.2 every time we measured an element to deduce whether it, along with the already measured set X, generated a larger subgroup. Once we've found a generating set for $A_p^{\perp}$, we apply Proposition 6.1.3 and produce a minimal generating set for A_p.

To wrap up we repeat the procedure for all the p-groups that make up G and then apply Proposition 6.1.1 to piece together their generating sets.

Running Time

According to Step 5 we must repeat the algorithm as many times as there are p-groups. Since the product of orders of the p-groups is equal to the order of G, there are at most $\log |G|$ of them. Steps 1 and 2 run in constant time but require implementing the Fourier transform as well as the function $f = M^{|A_p|}$. Step 3 needs to be repeated an expected at most $\log |G|$ number of times and Step 4 involves using the Propositions, which also run in time polynomial in $\log |G|$. The total running time then remains polynomial in $\log |G|$.

6.1.2 Semi-Structured Random Algorithm

Using Simon's Hidden Subgroup Problem as a tool in the solution to our problem might not be desirable because it involves using the Fourier Transform. The Fourier transform can't be implemented exactly for an arbitrary abelian group and must instead be estimated. If we wish to eliminate our dependence on it, and still calculate the generating sets of each p-group as well, there is another choice. The following algorithm works similarly to the one above, but instead of employing Simon's Hidden Subgroup Problem to extract generators for each p-group it instead uses the addition operator to directly sample from A_p. The advantage is the loss of dependency on the Fourier transform and the disadvantage is the heavier reliance on the addition operator.

Step 1 Take a n-qubit just large enough to represent all the elements in G. Set $N = 2^n$ and start off with the state $|0\rangle|e\rangle$ where $|e\rangle$ is the identity element of

the group. Apply the Hadamard transform to the first qubit to end up with

$$|\psi\rangle = \sum_{g=0}^{N-1} |g\rangle|e\rangle.$$

Step 2 Apply the operator $M^{\frac{|G|}{|A_p|}}$ to get the superposition

$$\sum_{g=0}^{N-1} |g\rangle|g^{\frac{|G|}{|A_p|}}\rangle$$

and measure the first n-qubit to obtain a random element of A_p, adding it to the set X.

Step 3 Apply the classical algorithm described by Proposition 6.1.2 to set X. If X is still too small (according to the cyclic decomposition of G) return to Step 1.

Step 4 Repeat the above process for all p-subgroups to find the generating sets for each. Then use the procedure described in Proposition 6.1.1 to find a minimal generating set for G.

The algorithm starts off with the standard superposition of all elements. You might note that some of the elements in the superposition aren't members of G, but that is of no concern as the overhead in running time is at most by a constant factor of 2. By applying $M^{\frac{|G|}{|A_p|}}$ to the superposition, we end up with a superposition of elements which are just in A_p. Just as in the first algorithm, we proceed to measure these elements, and in expected $\log|G|$ running time we come up with a generating set for A_p.

Running Time

Step 1 is easy to implement and takes constant time. Note that the Hadamard transform is efficiently and accurately implemented in $\log N$ time and is thus much better than the more unwieldy Fourier transform. Steps 2 and 3 are just like the ones in the previous algorithm and so require time polynomial in $\log|G|$, whereas Step 4 is identical to Step 5 from the previous algorithm and also takes at most $\log|G|$ repetitions. The difference in running time will be in the constants and the fact that this algorithm calls for the implementation of $M^{\frac{|G|}{|A_p|}}$ whereas the previous algorithm only needs $M^{|A_p|}$.

6.1.3 Unstructured Random Algorithm

This particular algorithm is here simply to demonstrate the effectiveness of a naïve random search for a generating set. It is based on selecting elements randomly from G without any prior manipulation. It might at first seem that this algorithm in fact bypasses all mention of the addition operator M, but in fact it makes implicit use of it through Propositions 6.1.2 and 6.1.3, as well as G's breakdown into cyclic groups and the decomposition function ϕ. Eventually the set X of elements that have been selected so far will be large enough to generate all of G. Each iteration of the algorithm is as follows:

Step 1 Begin with a uniform superposition of all elements in G, select one element randomly by measurement, and add it to the set X. Alternatively, select n random elements at a time and add them all to X. This could improve the running time of the algorithm, but by at most the constant factor n, so we will just consider the case where $n = 1$.

Step 2 Apply the algorithm described by Proposition 6.1.2 to X. Use the resulting linearly independent set in place of X from now on.

Step 3 Apply the algorithm described by Proposition 6.1.3 to X. If the orthogonal subgroup of $\langle X \rangle$ is just $\{0\}$, then X is a linearly independent subset of G that generates G, hence we are finished. Otherwise, go back to Step 1.

Running Time

At Step 1, we wish to select an element of G that is not already in the subgroup $\langle X \rangle$. As long as $\langle X \rangle$ is a proper subgroup of G, the probability that this will happen is at least $1/2$. If it does happen, then $\langle X \rangle$ will be enlarged by a factor of at least 2. The number of times that $\langle X \rangle$ needs to double in size is $\log |G|$, hence the expected number of times that Step 1 needs to be repeated is $O(\log |G|)$. Since the classical algorithms from Propositions 6.1.2 and 6.1.3 are polynomial in time $\log |G|$, the overall algorithm is polynomial in $\log |G|$.

6.2 Torsion-Free Abelian Group

Before presenting our algorithm for finding the generators of a Torsion-Free Abelian Group we first prove some results in probability which are then useful in discussing the efficacy of our algorithm.

6.2.1 Preliminaries

Given an event A, we write $P(A)$ to be the probability that event A occurs. We also write $(a_1, a_2, \ldots, a_n)$ to denote the greatest common divisor of $a_1, a_2, \ldots, a_n$.

Theorem 6.2.1. *For $a, b \in \mathbb{Z}$ we have $P\big((a,b)=1\big) = \frac{6}{\pi^2} = \zeta(2)^{-1}$. In general for $a_1, \ldots, a_n \in \mathbb{Z}$ we have $P\big((a_1, \ldots, a_n)=1\big) = \zeta(n)^{-1}$.*

Proof. Consider the probability that the numbers $a_1, a_2, \ldots, a_n$ are divisible by a prime p. Since the numbers picked are independent of each other, this is simply $\frac{1}{p^n}$ and so the probability that at least one of these numbers is not divisible by p is $1 - \frac{1}{p^n}$. If p and q are different primes, not being divisible by q and not being divisible by p are independent events, and therefore not being divisible by both p and q is just the product of the individual probabilities. We may then conclude that

$$P\big((a_1, \ldots, a_n)=1\big) = \prod_{p \text{ prime}} (1 - \frac{1}{p^n})$$

Dropping the subscript 'p prime' for convenience we then have

$$P\big((a_1, \ldots, a_n)=1\big)^{-1} = \prod \left(1 - \frac{1}{p^n}\right)^{-1} = \prod \frac{1}{1 - \frac{1}{p^n}} \tag{6.3}$$

$$= \prod \sum_{i=0}^{\infty} \left(\frac{1}{p^i}\right)^n = \sum_{k=1}^{\infty} \frac{1}{k^n} = \zeta(n). \tag{6.4}$$

So then $P\big((a_1, \ldots, a_n)=1\big) = \zeta(n)^{-1}$. □

We may think of picking numbers as picking 1×1 matrices and asking for the probability that their determinants are relatively prime. In the following theorems we will be discussing randomly chosen matrices from $M_n(\mathbb{Z})$. What we mean by this is that for any entry a of this matrix, any modulus m, and any integer k with $0 \leq k < m$, the probability that a is congruent to $k \pmod m$ is always $1/m$. This condition can be achieved if we bound a and then take the limit as that bound tends

to infinity. We would like to extend the above result to $n \times n$ matrices, but first we need a lemma.

Lemma 6.2.2. *Given $A \in M_n(\mathbb{Z})$ the probability that $\det A \neq 0 \pmod p$ for p a prime is*

$$\prod_{i=1}^{n} \left(1 - \frac{1}{p^i}\right).$$

Proof. We proceed by counting possibilities. For the first row of our matrix we have p^n -1 possibilities where we exclude only the 0 vector. We will show that the number of possibilities for the i^{th} row of our matrix, where $i \geq 1$, given that we have already chosen all the prior rows is $p^n - p^i$.

Let c_j denote the j^{th} row, where $0 \leq j < i$. Now it follows that the i^{th} row cannot be any of the vectors of the form $\sum_{k=0}^{i-1} a_k c_k$ where each of the $a_k \in \mathbb{Z}_p$. Since $\det A \neq 0 \pmod p$ we have that all the c_k are linearly independent. Since p is a prime all the linear combinations of c_k are distinct, and therefore there are exactly p^i vectors in $\{\sum_{k=0}^{i-1} a_k c_k \mid a_k \in \mathbb{Z}_p\}$. It follows that we have $p^n - p^i$ choices for the i^{th} row, and therefore the total number of possible matrices such that $\det A \neq 0 \pmod p$ is $\prod_{i=0}^{n-1}(p^n - p^i)$. Since the total number of different matrices modulo p is p^{n^2} the probability that $\det A \neq 0 \pmod p$ is

$$\frac{\prod_{i=0}^{n-1}(p^n - p^i)}{p^{n^2}} = \prod_{i=0}^{n-1} \frac{p^n - p^i}{p^n} = \prod_{i=1}^{n} \left(1 - \frac{1}{p^{n-i}}\right) = \prod_{i=1}^{n} \left(1 - \frac{1}{p^i}\right).$$

□

We can now calculate the probability that a determinant of an $n \times n$ matrix is divisible by p, namely

$$D_n(p) = 1 - \prod_{i=1}^{n} \left(1 - \frac{1}{p^i}\right). \tag{6.5}$$

With that done we prove the equivalent of Theorem 6.2.1 for $n \times n$ determinants.

Theorem 6.2.3. *Given k matrices $A_1, A_2, \ldots, A_k \in M_n(\mathbb{Z})$ we have that*

$$P\big((\det A_1, \det A_2, \ldots, \det A_k) = 1\big) = \prod_{p \text{ prime}} (1 - D_n(p)^k)$$

Proof. It should be clear that just as $p|n$ and $q|n$ are independent events for $n \in \mathbb{Z}$, if p and q are distinct primes, then $p|\det A$ and $q|\det A$ are independent events for

	n=1	n=2	n=3	n=4	n=5	n=6	n=7	n=8	n=9	n=10
k=2	0.609	0.453	0.397	0.374	0.364	0.359	0.356	0.355	0.354	0.354
k=3	0.832	0.692	0.629	0.6	0.587	0.58	0.577	0.575	0.575	0.574
k=4	0.924	0.821	0.766	0.739	0.726	0.72	0.716	0.715	0.714	0.714
k=5	0.964	0.894	0.85	0.827	0.815	0.81	0.807	0.805	0.805	0.804
k=6	0.983	0.936	0.902	0.883	0.874	0.869	0.867	0.865	0.865	0.864
k=7	0.992	0.961	0.936	0.921	0.913	0.909	0.907	0.906	0.906	0.905
k=8	0.996	0.976	0.957	0.946	0.94	0.937	0.935	0.934	0.934	0.933
k=9	0.998	0.985	0.972	0.963	0.958	0.955	0.954	0.954	0.953	0.953
k=10	0.999	0.991	0.981	0.974	0.971	0.969	0.968	0.967	0.967	0.967
k=11	1.	0.994	0.987	0.982	0.979	0.978	0.977	0.977	0.977	0.976
k=12	1.	0.996	0.991	0.988	0.986	0.984	0.984	0.984	0.983	0.983
k=13	1.	0.998	0.994	0.992	0.99	0.989	0.989	0.988	0.988	0.988
k=14	1.	0.999	0.996	0.994	0.993	0.992	0.992	0.992	0.992	0.992
k=15	1.	0.999	0.997	0.996	0.995	0.995	0.994	0.994	0.994	0.994

Table 6.1: Probabilities

$A \in M_n(\mathbb{Z})$. Since $D_n(p)$ is the probability that a determinant is divisible by p, clearly $1 - D_n(p)^k$ is the probability that k determinants don't all share p as a factor. Using lemma 6.2.2, the probability that these k determinants don't all share any prime factor, or in other words, that they're relatively prime, is simply

$$\prod_{p \text{ prime}} \left(1 - D_n(p)^k\right) = \prod_{p \text{ prime}} \left(1 - \left(1 - \prod_{i=0}^{n-1} \left(1 - \frac{1}{p^i}\right)\right)^k\right) \tag{6.6}$$

□

Though we were not able to get useful and rigorous estimates for the lower bound on this probability, we did calculate some values up to a point where it seemed like the product was converging. The values in table 6.2.1 are evaluations of expression (6.6) up to 30 distinct primes. The table is arranged by number of matrices picked determined by the row, and number of dimensions determined by the column. If this is to be taken as an indication of the limiting values, then even at a small number of matrices picked, the probabilities are already very close to 1. Notice that the first column, since it's simply 1×1 matrices should look like $\zeta(n)^{-1}$.

We now come to why it was useful to consider the probability with which picking random matrices would produce relatively prime determinants.

Lemma 6.2.4. *Let $a_1, a_2, \ldots, a_n$ represent the standard basis vectors of $\mathbb{Z}^n$. Given n column vectors in $\mathbb{Z}^n$ whose determinant is k, the vectors $ka_1, ka_2, \ldots, ka_n$ are all in the subgroup generated by the n column vectors.*

Proof. Let A be the matrix composed of the n given column vectors. A given vector a is a linear combination of the given n column vectors if and only if there exists a column vector w such that $Aw = a$. Now let $\tilde{A}$ denote the adjunct of the matrix A; that is, the matrix such that $\tilde{A}A = (\det A)\, I$ where I is the identity matrix. From basic linear algebra we know that if $A \in M_n(\mathbb{Z})$ then $\tilde{A} \in M_n(\mathbb{Z})$, so we can solve for w_i in the equation $Aw_i = ka_i$ for $i = 1 \ldots n$, namely

$$\begin{aligned} Aw_i &= ka_i; \\ \tilde{A}Aw_i &= k\tilde{A}a_i; \\ kw_i &= k\tilde{A}a_i; \\ w_i &= \tilde{A}a_i. \end{aligned}$$

□

Theorem 6.2.5. *Adopt the notation of the previous lemma. If given m sets of n vectors $\{v_i\}_j$, for $i = 1 \ldots n$ and $j = 1 \ldots m$, such that the $n \times n$ matrices formed for each j have determinants $k_1, k_2, \ldots, k_m$ respectively, and $(k_1, k_2, \ldots, k_m) = 1$, then together the mn vectors generate all of $\mathbb{Z}^n$.*

Proof. By Lemma 6.2.4 we know that each set $\{v_i\}_j$ can generate $k_j a_l$ for $l = 1 \ldots n$. From basic number theory we also know that if $(k_1, k_2, \ldots k_m) = 1$ then there exists a linear combination such that $x_1k_1 + x_2k_2 + \cdots + x_mk_m = 1$ for $x_1, x_2, \ldots x_m \in \mathbb{Z}$. It is clear then that for all l the vector a_l lies in the space generated by $\{v_i\}_j$ and therefore this space is all of $\mathbb{Z}^n$. □

6.2.2 The Algorithm

First we introduce our assumptions, which are not all that different from the finite case of our problem. Say our abelian group is $\mathbb{Z}^n$. Then we assume that we're given a function ϕ that can break down an element into its components. We also assume that given two elements we are able to compute their sum. Since the group is infinite and therefore choosing an element randomly is impossible, we assume that we're given

a large subset of the group, $S \subset \mathbb{Z}^n$, that we know generates it. Our task is to find elements of this subset that form a minimal generating set. The algorithm once again relies on randomly choosing elements. As we will see the success and favorable running time of the algorithm depends on Theorems 6.2.3 and 6.2.5.

Step 1 Choose $m = kn$ random elements from S, for some integer k.

Step 2 Perform a variation of Gaussian elimination to reduce the set of m elements to a smaller set of n elements that span the same space.

Step 3 Measure the determinant of the n elements. If the determinant is ± 1 then we're done, otherwise, go back to Step 1 and pick more elements.

There are two crucial steps in the algorithm. The first is the ability to perform Step 2 above, and we describe that shortly, and the second is verifying that the running time, which is determined by Step 3, is reasonable in n. The motivation for the algorithm comes from Theorems 6.2.3 and 6.2.5, and the running time analysis depends on the probabilities derived therein. We now further describe the implementation of Step 2.

Our strategy is to place the m selected elements into an $n \times m$ matrix A based on the decomposition of each element that we can get from ϕ. Note that every column in this matrix represents one of the randomly chosen elements, and the rows represent the elements' components according to ϕ. Via the following theorems we will show how to reduce this matrix to one in upper triangular form A'. That is, all entries to the left of the rightmost diagonal will be 0, and the columns of A' should span exactly the same space as the columns of A. Since A' will have at most n nonzero columns, we will have completed Step 2.

Lemma 6.2.6. *If A is an $n \times m$ matrix, and B is an $m \times m$ matrix whose determinant is ± 1, then the columns of AB generate the same subgroup of $\mathbb{Z}^n$ as the columns of A.*

Proof. The subgroup of $\mathbb{Z}^n$ generated by the columns of any $n \times m$ matrix X is the set of all $\mathbb{Z}$-linear combinations of the columns, which is just the set $\{Xz \mid z \in \mathbb{Z}^m\}$. It is clear that $\{ABz \mid z \in \mathbb{Z}\} \subseteq \{Ay \mid y \in \mathbb{Z}\}$, just take $y = Bz$. However, the reverse inclusion is also true, because B^{-1} exists and has integer entries, so we can take $z = B^{-1}y$. □

We will make use of the following proposition, which allows us to insert zeroes into A.

Theorem 6.2.7 (Alex Eustis)**.** *If A is an $n \times m$ matrix and i, j are integers with $1 \leq i \leq n$ and $1 \leq j \leq m$, then there exists an $m \times m$ matrix B_{ij} such that:*

(i) $\det B = 1$.

(ii) $(AB)_{ij} = 0$.

(iii) Multiplication on the right by B does not alter any of the columns of A the j^{th} and $(j+1)^{st}$ columns.

(iv) If the entries $A_{i'j}$ and $A_{i'(j+1)}$ are both zero for some i' with $1 \leq i' \leq n$, then $AB_{i'j}$ and $AB_{i'(j+1)}$ are also zero.

Proof. Let $A_{ij} = a$ and $A_{i(j+1)} = b$. If $a = 0$ then just let $B = I$. Otherwise, let $g = \gcd(a, b)$ (hence we know that $g \neq 0$), and let $u, v \in \mathbb{Z}$ be integers such that $au + bv = g$. We know that u and v can be determined efficiently using Euclid's Algorithm.

We define B using block notation in the diagram below, where I_k represents the $k \times k$ identity matrix.

$$B = \left(\begin{array}{c|cc|c} I_{j-1} & 0 & & 0 \\ \hline 0 & b/g & u & 0 \\ & -a/g & v & \\ \hline 0 & 0 & & I_{m-j-1} \end{array}\right)$$

It should be noted that $\det B = \frac{bv}{g} + \frac{au}{g} = 1$, and that right multiplication of A by B only changes the j^{th} and $(j+1)^{st}$ columns of A; it leaves the other columns unchanged. In fact, the matrix product AB will give the following

$$AB = \begin{pmatrix} \cdots & \vdots & \vdots & \cdots \\ & a & b & \\ \cdots & \vdots & \vdots & \cdots \\ & 0 & 0 & \\ \cdots & \vdots & \vdots & \cdots \end{pmatrix} \left(\begin{array}{c|cc|c} I_j & 0 & & 0 \\ \hline 0 & b/g & u & 0 \\ & -a/g & v & \\ \hline 0 & 0 & & I_{m-j-1} \end{array}\right) = \begin{pmatrix} \cdots & \vdots & \vdots & \cdots \\ & 0 & g & \\ \cdots & \vdots & \vdots & \cdots \\ & 0 & 0 & \\ \cdots & \vdots & \vdots & \cdots \end{pmatrix}$$

where $\cdots$ represents arbitrary integers that do not concern us. It is clear then that with B defined as above all four conditions stated in the theorem are satisfied. $\square$

To complete the process called for by Step 2, let A be the $n \times m$ matrix of sampled elements of S, and let B_{ij} denote the $m \times m$ matrix defined above that annihilates the entry A_{ij}. We can put A into upper right triangular form by computing the following matrix product:

$$A' = A(B_{n1}B_{n2}\dots B_{n(m-1)})(B_{(n-1)1}\dots B_{(n-1)(m-2)})\dots(B_{11}\dots B_{1(m-n)}) \qquad (6.7)$$

That is, we annihilate the whole bottom row from left to right except for the rightmost entry. Then we annihilate the next row up except for the rightmost two entries, and continue in this manner up to the top row, which we annihilate except for the rightmost n entries. Parts (iii) and (iv) of Theorem 6.2.7 guarantee that none of the zeros created by the previous B_{ij}'s will be disturbed by the ones later on. Therefore A' is a matrix whose columns generate the same subgroup of $\mathbb{Z}^n$ as the columns of A, and only the rightmost n columns of A' can be nonzero. If we identify A' with its rightmost n columns, then we can easily compute its determinant as the product along its main diagonal, since it is in upper triangular form. If $\det A' = \pm 1$, then the columns of A' form a minimal generating set for $\mathbb{Z}^n$, and we are done. If any other determinant is obtained, we know that the m elements that we began with do not generate $\mathbb{Z}^n$, so we must go back and choose some more.

Running Time

Each multiplication by B_{ij} requires finding the greatest common divisor of two integers, which runs in $O(\log^2 N)$ if the integers are bounded by N, and also requires four multiplications of columns, which amounts to $4n$ integer multiplications, which runs in $O(n \log^2 N)$ time, once again if N is a bound for the size of the integers. The number of B_{ij}'s required is less than mn, and recall that $m = kn$ where k is a constant. So we must multiply $O(n^2)$ number of B_{ij}'s to determine A', and therefore the whole elimination algorithm runs in time $O(n^3 \log^2 N)$. Though we don't have a good bound yet, Theorem 6.2.3 seems to suggest that the probability of choosing matrices with relatively prime determinants doesn't decrease fast with n and is perhaps related logarithmically. Even a probability decrease as high as $\frac{1}{n}$ would seem reasonable and would leave the running time at $O(n^4 \log^2 N)$.

6.3 Ideas for Further Research

The first outstanding problem would be to calculate an efficient lower bound for the probability of picking relatively prime determinants, equation 6.6, as that would determine a running time order of growth for our algorithm. Further research could be done on ways to make our algorithms deterministic. As it stands all of the algorithms have good expected running times, yet because they're all based on random choosing of elements, there's always a chance that a particular case might take a very long time. Another problem to consider is constructing an efficient algorithm for finding generators for a finitely-generated abelian group. This was a problem we worked on but did not get sufficient results on. One might attempt first to find the generators for the torsion free part of the group using a method similar to our modification of Gaussian elimination, and then to employ the finite group algorithms on the finite group components of elements to get generators for the torsion group. However since the sample set of elements given is so large, in fact most likely orders of magnitude larger than the order of the torsion group, the finite group algorithms would not be efficient. Either a modification or an altogether different approach must then be used to extract the generators for the torsion group, at which point the torsion and torsion free group generators may be combined.

Chapter 7

Conclusion

We have described numerous remarkable results regarding quantum computers. The circuit model for quantum computation has been rigorously defined and employed in the construction of Shor's factoring and Grover's search algorithms. Fundamental limitations as well as information processing feats of quantum computers have also been detailed. All of these results and others were placed in their proper settings during a discussion of complexity theory, and the Strong Church-Turing thesis was mentioned as the guiding light for research. Finally, the paper concluded with research on the problem of finding generators for an abelian group and presented several algorithms to solve that and a similar problem.

The techniques presented in Shor's and Grover's algorithms remain the most widely used in designing quantum algorithms. The reason so much focus has been placed on these algorithms is that it is a generally hard problem to design quantum computing algorithms. This is not necessarily because quantum computers are more complicated to work with than classical ones, but rather because a quantum algorithm is only interesting if it runs faster than any known classical counterpart. In fact, generally, even polynomial time increases are not interesting unless they are to already fast, fundamental problems, like an unstructured search. Most of the emphasis in quantum computing, especially from a complexity theory point of view, is bridging the gap between classically tractable and intractable problems.

As was seen however, complexity theory has a lot left to accomplish. Results which seem as though they should be obvious, for example $\mathbf{P} \subset \mathbf{PSPACE}$, prove to be incredibly subtle. It might be a long time before quantum computers are proved

to be theoretically more powerful, in terms of efficient algorithms, than probabilistic Turing machines. Practically speaking however, the observed differences are real. If a quantum computer could be built right now and Shor's algorithm implemented, it would have a tremendous impact on encryption algorithms regardless of whether we have a theoretical result that says that factoring is in **BPP** or not. As we have said, Shor's algorithm isn't the only advantage to quantum computers. Grover's algorithm can be used to speed up thousands of commonly used algorithms, and quantum simulation would become tractable. Physical realizations of quantum computers are battling with the noise introduced when one tries to operate on more than a few qubits, but error correcting codes have achieved promising results of how to increase robustness and avoid the fate of the analog computer.

As time goes on more and more experimental methods for building quantum computers are found and improved upon. Though theoretical complexity results showing the superiority of quantum computers would cause quite a stir in the community and have, deep, long lasting effects on the notion of what is efficiently computable, the real impact of quantum computers is already here. Nielsen and Chuang [NC00, p. 46-50] give a good overview of small, medium, and large scale applications of quantum computing that could be pursued. Such applications as quantum state tomography, distribution of information through quantum teleportation, speeding up of algorithms via Grover's algorithm, and quantum simulation are just some of the already theoretically realizable achievements of quantum computing. None of these applications can at present be resolved with efficient classical algorithms, and, in fact, the lack of said algorithms demonstrates that these problems are hard classically, and even if theoretically possible, it will be a long time before efficient classical algorithms may be developed. Indeed, we stand to benefit greatly from quantum computers.

Bibliography

BBBV97. Charles H. Bennett, Ethan Bernstein, Gilles Brassard, and Umesh Vazirani. Strengths and weaknesses of quantum computing. *Special issue on Quantum Computation of the Siam Journal of Computing*, Oct. 1997. Also available as `http://arxiv.org/abs/quant-ph/9701001`.

BH97. G. Brassard and P. Høyer. An exact quantum polynomial-time algorithm for simon's problem. In *Proceedings of the Fifth Israeli Symposium on Theory of Computing and Systems - ISTCS*, pages 12–23, 1997. Also available as `http://arxiv.org/abs/quant-ph/9704027`.

BV97. Ethan Bernstein and Umesh Vazirani. Quantum complexity theory. *SIAM J. Comput.*, 26(5):1411–1473, 1997. Also available as `citeseer.ist.psu.edu/bernstein97quantum.html`.

BW92. Charles H. Bennet and Stephen J. Wiesner. Communication via one- and two-particle operators on einstein-podolsky-rosen states. *Physical Review Letters*, 69(20):2881–2884, 1992.

Cle99. Richard Cleve. An introduction to quantum complexity theory. `http://arxiv.org/abs/quant-ph/9906111`, 1999.

Coo71. S. Cook. The complexity of theorem-proving procedures. In *Proc. 3rd ACM Symp. Theory of Computing*, pages 151–158, 1971.

Edm65a. J. Edmonds. Maximum matchings and a polyhedron with 0,1-vertices. *Journal of Research at the National Bureau of Standards (Section B)*, pages 125–130, 1965.

Edm65b. J. Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.

Fey82. R. Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 21, 1982.

FH02. Lance Fortnow and Steve Homer. A short history of computational complexity. `http://citeseer.ist.psu.edu/fortnow02short.html`, 2002.

Gas02. William I. Gasarch. The $P =?NP$ Poll. `www.cs.umd.edu/~gasarch/papers/poll.ps`, 2002.

Gud03. Stan Gudder. Quantum computation. *Mathematical Association of America Monthly*, 110(3):181–201, 2003.

Hir04. Mika Hirvensalo. *Quantum Computing*. Springer, Berlin, 2004.

HS65. J. Hartmanis and R. Stearns. On the computational complexity of algorithms. *Transactions of the American Mathematical Society*, 117:285–306, 1965.

HS99. F. Hennie and R. Stearns. Two-tape simulation of multitape turing machines. *Journal of the ACM*, 13(4):1364–1396, August 1999.

HW68. G. H. Hardy and E. M. Wright. *An Introduction to The Theory of Numbers*. Oxford University Press, fourth edition, 1968.

Kar72. R. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–104. Plenum-Press, New York, 1972.

Knu97. Donald Knuth. *The Art of Computer Programming*, volume 2. Addison-Wesley, third edition, 1997.

Lan90. Serge Lang. *Undergraduate Algebra*. Springer, 1990.

Lan94. Serge Lang. *Introduction to Linear Algebra*. Springer, 1994.

NC00. Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.

Pit00. Arthur O. Pittenger. *An Introduction to Quantum Computing Algorithms.* Birkhäuser, 2000.

Rig04. Gustavo Garcia Rigolin. Superdense coding using multipartite states. `http://arxiv.org/abs/quant-ph/0407193`, 2004.

Sho97. Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J.SCI.STATIST.COMPUT.*, 26:1484, 1997. Also available as `http://arxiv.org/abs/quant-ph/9508027`.

Sto76. L. Stockmeyer. The polynomial-time hierarchy. *Theor. Computer Science*, 3:1–22, 1976.

VBE96. Vlatko Vedral, Adriano Barenco, and Artur Ekert. Quantum networks for elementary arithmetic operations. *Physical Review Letters*, 54(1):147–153, 1996. Also available as `http://arxiv.org/abs/quant-ph/9511018`.

Yao90. A. Yao. Coherent functions and program checkers. In *Proceedings of the 22nd ACM Symposium on the Theory of Computing*, pages 84–94, New York, 1990. ACM.