SUMMER OF SCIENCE 2021

Final Report

Quantum Computing and Quantum Information

Adish Shah

Mentor: Anaida Ali
Roll No: 200020012

Contents

Chapter 1

Introduction

Quantum computation and quantum information is the study of information processing tasks which can be accomplished using quantum mechanical systems. At the face of it, this sounds pretty simple and obvious. But like many simple and profound ideas, it was a long time before anybody thought of doing information processing using quantum mechanical systems.

To see why this is the case, we must look in turn at each of the fields which have contributed fundamental ideas to quantum computation and quantum information – quantum mechanics, computer science, information theory, and cryptography.

In this Summer of Science report, I attempt to summarise everything that I've covered in quantum computing till now. I mainly followed the book **Quantum Computation and Quantum Information** by **Nielsen and Chuang**. I will begin by going through the mathematical preliminaries which includes an introduction to the linear algebraic treatment of quantum mechanics through which we will explore quantum computation.

Chapter 2

Mathematical Preliminaries

Our treatment of quantum computation heavily uses mathematical preliminaries. Hence, I will summarise some key theorems and notations that I will be using.

2.1 Vectors and Vector Spaces

The most important structure in linear algebra is a vector which is represented as $|\psi\rangle$. This vector resides in a vector space that satisfies some axioms that give this structure its core defining properties. This representation of a vector is famously known as the *bra-ket* notation and the $|.\rangle$ is used to represent a vector. The same vector in the vector space $\mathbb{C}^n$ is conveniently represented as a column vector.

We will be generally working in the vector space $\mathbb{C}^n$ over the field $\mathbb{C}$. A set of vectors $|\psi_1\rangle, |\psi_2\rangle \cdots |\psi_n\rangle$ is said to be *linearly-independent* if for any set of coefficients a_i,

$$\sum_{i=1}^{n} a_i |\psi_i\rangle = 0 \implies a_i = 0 \, \forall 1 \leq i \leq n$$

A set of vectors $|\psi_i\rangle$ is said to be a spanning set of a vector space $\mathbf{W}$ if any vector in $\mathbf{W}$ can be represented as a linear combination of that set. Note that such a set may be non-finite however we will be dealing with finite dimensional vector spaces only. If this set is linearly independent then this set is called a *basis* of that vector space. The cardinality of all bases are same and this value is termed as the *dimension* of that vector space.

2.2 Linear Operators and Matrices

A linear operator A from vector space $\mathbf{V}$ to $\mathbf{W}$ is a function mapping vectors from $\mathbf{V}$ to $\mathbf{W}$ satisfying linearity, that is:

$$A(\sum_i a_i |\psi_i\rangle) = \sum_i a_i A |\psi_i\rangle$$

We can then define compositions of operators, BA as function that first operates A and then operates B.

It is easy to see that matrix multiplication is a linear operator. Conversely any linear operator has a matrix representation. To see this, let $A : V \to W$ be a linear operator. Let $|v_i\rangle$ be an ordered basis (basis formed by fixing order of its elements, in this case by enforcing an order restriction on $|v_i\rangle$.) of V and $|w_j\rangle$ be an ordered basis of W.

Then there exists complex numbers $A_{i,j}$ such that:

$$A |v_i\rangle = \sum_j A_{i,j} |w_j\rangle$$

If we represent any vector in V by its coordinate vector representation, that is form a column vector whose elements with respect to an ordered basis are the scalar coefficients along the basis elements and multiply it by the matrix formed by $A_{i,j}$ we get the coordinate vector representation of the output vector in the ordered basis $|w_j\rangle$.

2.3 Inner Products

An inner product of a vector space $\mathbf{V}$ is defined as a function from $\mathbf{V} \times \mathbf{V}$ to $\mathbf{C}$. We will denote the inner product of $|v\rangle$ and $|w\rangle$ as $(|v\rangle, |w\rangle)$ or as $\langle v|w\rangle$. The inner product satisfies the following properties:

1. $\langle v|w\rangle = (\langle w|v\rangle)^*$
2. $\left\langle v \middle| \sum_j a_j |w_j\rangle \right\rangle = \sum_j a_j \langle v|w_j\rangle$
3. $\langle v|v\rangle \geq 0$ with equality *iff* $v = 0$

We can now define the *length* or *norm* of a vector $\||v\rangle\| = \sqrt{\langle v|v\rangle}$. Two vectors are said to be orthogonal if their inner product is 0. We now define an *orthonormal* basis as a basis $|v_i\rangle$ whose elements have unit norm and they are pairwise orthogonal, that is for $i \neq j \, \langle v_i|v_j\rangle = 0$. It can be proven that every finite dimensional vector space has an orthonormal basis. This is left as an exercise. The procedure for determining this orthonormal set from any basis is termed as the *Gram-Schmidt* orthogonalisation process.

This leads to a simple representation of the inner products of two vectors in matrix form. Let $|w\rangle = \sum_i w_i |i\rangle$ and $|v\rangle = |v\rangle = \sum_j v_j |j\rangle$ be two vectors written with the same orthogonal basis. Then $\langle w|v\rangle = (\sum_i w_i |i\rangle, \sum_j v_j |j\rangle) = \sum_i w_i^* v_i$ which we can nicely write in matrix form as

$$\begin{bmatrix} v_1^* & v_2^* & \cdots & v_n^* \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}$$

One use of this can be seen as follows:

Let $|i\rangle$ be an orthonormal basis of a vector space $\mathbf{V}$. Then for any vector $|v\rangle$ in this vector space, it is easy to see that

$$(\sum_i |i\rangle \langle i|) |v\rangle = |v\rangle$$

which in turn implies that

$$\sum_i |i\rangle \langle i| = I_v$$

This is termed as the completeness relation.

Using this we can easily get the matrix representation of an operator $A : \mathbf{V} \to \mathbf{W}$. Let $|v_i\rangle$ be an orthonormal basis for $\mathbf{V}$ and $|w_j\rangle$ be an orthonormal basis for $\mathbf{W}$

$$A = I_W A I_V = \sum_{i,j} |w_j\rangle \langle w_j| A |v_i\rangle \langle v_i|$$

The terms $\langle w_j| A |v_i\rangle$ are nothing but the entries of the matrix corresponding to this transformation written in coordinate vector form.

2.4 Eigenvalues and Eigenvectors

An eigenvector for a linear operator A is a non-zero vector $|v\rangle$ such that $A|v\rangle = v|v\rangle$ for some complex number v. This number is called the eigenvalue corresponding to this eigenvector and we will refer to both the eigenvalue and the eigenvalue by the same letter.

A sufficient and necessary condition for a number λ to be an eigenvalue is

$$\det |A - \lambda I| = 0$$

We will term the subspace spanned by eigenvectors corresponding to the same eigenvalue as the eigenspace corresponding to that eigenvalue. Now we introduce diagonalizable operators. An operator is said to be diagonalizable on a vector space if there exists an orthonormal basis composed of eigenvectors of that operator. It is left as an exercise to show that such an operator A can be represented as:

$$A = \sum_i \lambda_i |i\rangle \langle i|$$

where $|i\rangle$ corresponds to the orthonormal eigenvectors and λ_i are the corresponding eigenvalues. We will now look at some conditions under which operators are diagonalizable. This is of importance to us as we will be using the diagonal representation of the operator many times later on.

2.5 Adjoints and Hilbert Spaces

For any linear operator A it can be proven that there is a unique linear operator $A^\dagger$ satisfying

$$(|w\rangle, A|v\rangle) = (A^\dagger |w\rangle, |v\rangle)$$

This operator is termed as the adjoint of A. It can be shown that the matrix representation of $A^\dagger$ is obtained by taking the complex conjugate and then the transpose of A or $A^\dagger = (A^*)^T$.

Definition 2.5.1. An operator is said to be **Hermitian** if $A^\dagger = A$.

We proceed to show an example of an useful hermitian operator, the Projector. Let $\mathbf{W}$ be a d dimensional subspace of $\mathbf{V}$ which is k dimensional. Let $|v_1\rangle \cdots |v_d\rangle$ be an orthonormal basis of $\mathbf{W}$. By Gram Schmidt process we can extend this to an orthonormal basis of $\mathbf{V}$. The Projector $P : \mathbf{V} \to \mathbf{W}$ is defined as:

$$P = \sum_{i=1}^{d} |v_i\rangle \langle v_i|$$

Informally this projector projects vectors from $\mathbf{V}$ to $\mathbf{W}$ as it only leaves those components of the vector that are along the basis vectors which are in $\mathbf{W}$. We leave some trivial exercises corresponding to the Projector below that can be done by following the definitions.

Definition 2.5.2. An operator is A said to be **normal** if $A^\dagger A = AA^\dagger$.

The following theorem is incredibly useful and we omit its proof as it is rather involved:

Theorem 2.5.1 (Spectral Theorem)**.** An operator A is diagonalizable if and only if it is normal.

Now we talk about a special type of normal operators, unitary operators. These operators are of particular interest to us as all single quantum gates can be represented as unitary operators.

Definition 2.5.3. An operator is A said to be **unitary** if $A^\dagger A = I$.

A special class of hermitian operators is of particular interest to us. These operators called positive operators are useful as they come up in the polar and singular decompositions which we will discuss shortly.

Definition 2.5.4. An operator is A said to be **positive** if $(|v\rangle, A|v\rangle) \in \mathbb{R}^+$ for all $|v\rangle$

2.6 Operator Functions

It becomes useful for us to define functions on operators that are normal. Formally if A is a normal operator and its diagonal representation is as follows:

$$A = \sum_i v_i |v_i\rangle \langle v_i|$$

then we define

$$f(A) = \sum_i f(v_i) |v_i\rangle \langle v_i|$$

It is easy to see that $f(A)$ is uniquely determined which allows us to define things like square roots of positive operators, exponentials of normal operators etc.

We briefly mention that the notion of trace for a matrix can be extended for operators. This is because the trace satisfies

$$tr(AB) = tr(BA)$$

for matrices A, B. Suppose $A^{'}$ is one matrix representation of A. Then it is related to another matrix representation by $A^{''}$ by

$$A^{'} = PA^{''}P^{-1}$$

for some change of basis matrix P. Clearly by the previous result $tr(A^{'}) = tr(A^{''})$ and so trace of an operator is independent of its matrix representation.

2.7 Polar and Singular Value Decompositions

We omit the proof of the polar decomposition as it is rather involved. The singular value decomposition follows from the polar decomposition.

Theorem 2.7.1. Polar Decomposition For any linear operator A on a vector space $\mathbf{V}$ there exists unitary U and positive J, K such that

$$A = UJ = KU$$

where

$$J = \sqrt{A^\dagger A}, K = \sqrt{AA^\dagger}$$

Theorem 2.7.2. Singular Decomposition For any square matrix A there exists unitary matrices M, N and diagonal matrix D with positive entries such that

$$A = MDN$$

Proof. By the polar decomposition, there exists unitary U and positive J such that

$$A = UJ$$

As J is positive, it is diagonalisable, so

$$J = T^\dagger DT$$

where the entries of D are the eigenvalues of J which are positive and T is unitary. Setting $M = UT^\dagger$ and $N = T$ completes the proof. □

We end the mathematical preliminaries at this stage and move onto the basics of quantum mechanics which will be useful in our study of quantum computation.

Chapter 3

Introduction to Quantum Mechanics

Quantum mechanics is a mathematical framework for the development of physical theries. On its own quantum mechanics doesn't tell you what laws a physical system must obey, but it does provide a mathematical and conceptual framework for the development of such laws. In the next few sections we give a complete description of the basic postulates of quantum mechanics. These postulates provide a connection between the physical world and the mathematical formalism of quantum mechanics. We'll see how they can be useful in our study of quantum computation.

3.1 Postulates of Quantum Mechanics

3.1.1 The State Space

The first postulate sets up the arena in which quantum mechanics takes place which is the Hilbert space.

Postulate 1. Associated to any isolated physical system is a complex vector space with inner product (that is, a Hilbert space) 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.

This postulate states that any system can be described as a vector spanned by some basis elements. Knowing a basis for our vector space allows us to therefore express the entire system at any point of time in terms of these elements and the transformation of this representation with time is sufficient for us to describe how the system evolves with time.

However, quantum mechanics does *not* tell us, what the state space or state vector is , for a given physical system. It merely states its existence. Figuring that out for a specific system is a challenging problem. For this we need to develop additional quantum field theories like quantum electrodynamics (often known as QED) which describes how light and atoms interact.

Let us take a simple example of a quantum mechanical system, which is the *qubit*. Classical computers always process data in forms of the bits 0 and 1. But any bit at any point of time can either be 0 or 1. Quantum Computers however allow superposition of these bits.

A qubit can therefore be one of the possible states, that is an element of the state space. The basis elements of this state space are taken as 0 and 1, which we write in vector form as $|0\rangle$ and $|1\rangle$. We can therefore write any qubit $|\psi\rangle$ as

$$|\psi\rangle = a\,|0\rangle + b\,|1\rangle\,,$$

where a and b are complex numbers. The postulate dictates that $|\psi\rangle$ is a unit vector. This ensures that $|a|^2 + |b|^2 = 1$. This must hold even after any transformation is done on the current state vector.

Note that our choice of using $|0\rangle$ and $|1\rangle$ as the basis is arbitary. We can use other bases as well. The basis we use is referred to as the computational basis.

3.1.2 Evolution

How exactly does the state, $|\psi\rangle$, of a quantum mechanical system change with time? The following postulate give a prescription for the description of such state changes.

Postulate 2. The evolution of a *closed* quantum system is described by a *unitary transformation*. That is, the state $|\psi\rangle$ of the system at time t_1 is related to the state $\left|\psi^{'}\right\rangle$ of the system at time t_2 by a unitary operator U which depends only on the times t_1 and t_2,

$$\left|\psi^{'}\right\rangle = U\,|\psi\rangle$$

In our study of quantum computation we will be using various gates. We can consider a gate as a machine that takes in a quantum state and churns out another at a later time. The above postulate thus allows us to model all gates in the form of unitary transformations. However, it does not tell us which unitary operators can be used to describe the evolution of a system.

It turns out that for quantum gates on a single qubit, any unitary operator can be used to describe evolution via a suitable quantum gate. We describe some gates here in brief. Consider a quantum NOT gate. This must take $|0\rangle$ to $|1\rangle$ and vice-versa. This operator is simply:

$$X = |0\rangle\langle 1| + |1\rangle\langle 0|$$

which in matrix form is simply $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$

Similarly we have the phase flip gate Z which takes keeps $|0\rangle$ the same and takes $|1\rangle$ to $-|1\rangle$. Another useful quantum gate is the *Hadamard Gate*,

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

3.1.3 Measurement

We postulated that closed quantum systems evolve according to unitary evolution. The evolution of systems which don't interact with the rest of the world is all very well, but there must also be times when the experimentalist and their experimental equipment – an external physical system in other words – observes the system to find out what is going on inside the system, an interaction which makes the system no longer closed, and thus not necessarily subject to unitary evolution. To explain what happens when this is done, we introduce Postulate 3, which provides a means for describing the effects of measurements on quantum systems.

Postulate 3. Quantum measurements are described by a collection $|M_m\rangle$ of *measurement operators*. These are 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 $|\psi\rangle$ immediately before the measurement, then the probability that result m occurs is given by

$$p(m) = \langle\psi|\, M_m^\dagger M_m\,|\psi\rangle$$

and the state of the system after the measurement is

$$\frac{M_m\,|\psi\rangle}{\sqrt{\langle\psi|\, M_m^\dagger M_m\,|\psi\rangle}}$$

The measurement operators satisfy the *completeness equation*,

$$\sum_m M_m^\dagger M_m = I$$

.

The completeness equation expresses the fact that probabilities sum to one:

$$1 = \sum_m p(m) = \sum_m \langle\psi| M_m^\dagger M_m |\psi\rangle$$

This postulate fundamentally describes one of the major features of quantum mechanics, that measurement changes the system. After measuring the state vector collapses into another state.

We now talk about a different way of talking about measurements, the projective measurement operators. Suppose that the act of measurement is denoted by an operator M acting on our state space. Note that this can be done only when your system is closed, or when unitary dynamics can be applied to it (energy is conserved). The act of measurement may change this as external influence is applied.

However assuming energy is conserved than correspondingly M will be unitary and hence will have a spectral decomposition. Moreover as we will be measuring "real" values the eigenvalues will be real and thus M will be hermitian. Let m refer to the eigenvalues of this observable. Then

$$M = \sum_m m P_m$$

where P_m is the projector onto the eigenspace corresponding to the eigenvalue m. The act of observing the observable M will collapse the state vector into one of the eigenstates. Notice that this is similar to the previous postulates in the special case when our measurement operators are the projectors onto the eigenstates, as if $M_m = P_m$ then

$$P_m = M_m^\dagger M_m$$

Hence by the previous postulate the probability of getting a value m is

$$p(m) = \langle\psi| P_m |\psi\rangle$$

and the state after the measurement is

$$\frac{P_m |\psi\rangle}{\sqrt{p(m}}$$

The average value of the measurement is then simply

$$E(m) = \sum_m m p(m) = \sum_m m \langle\psi| P_m |\psi\rangle = \langle\psi| M |\psi\rangle$$

However projective measurements are not all type of measurements. It can be shown that projective measurements along with unitary transformations can account for all measurment operators. We can however make the math corresponding to postulate 3 simpler by invoking a mathematical tool called POVM measurements.

3.1.4 POVM Measurement

Notice that in Projective measurement the measurement operators were the projectors themselves. As projectors satisifed $P^\dagger P = P$ the math corresponding to the probabilities becomes simple. Notice that the only place where we need M_m is to get the state of the system after the measurement. If we do not need or care about this, we can define a new set of operators

$$E_m = M_m^\dagger M_m$$

Then corresponding to E_m there is an eigenvalue m which could be obtained after the measurement. The probability simply becomes

$$p(m) = \langle\psi| E_m |\psi\rangle$$

and the completeness relation implies

$$\sum_m M_m^\dagger M_m = \sum_m E_m = I$$

The set of operators $\{E_m\}$ is termed as *POVM* and each E_m is called a *POVM* element. A POVM is sufficient to obtain the probabilities of each measurable value and makes it more elegant reducing the need for adjoints. We have shown that for every set of measurement operators there is a corresponding set of POVMs.

Note that POVMs and general measurement operators are more general than projective measurements as we do not comment on the nature of the measurement operators. However it can be proven using composite quantum systems that projective measurements along with unitary operators are sufficient to describe any general measurment. We reserve this proof and discussion for a later time.

We briefly mention the final postulate which deals with composite quantum systems. We will revisit this later when we deal with tensor products and partial traces.

3.2 Composite Quantum Systems

Postulate 4. The state space of a composite physical system is the tensor product of the state spaces of the component physical systems. Moreover, if we have systems numbered 1 thorugh n, and system number i is prepared in the state $|\psi_i\rangle$, then the joint state of the total system is

$$|\psi_1\rangle \otimes |\psi_2\rangle \otimes \cdots |\psi_n\rangle$$

The symbol $\otimes$ is used for the tensor product. Essentially tensor products are used to build composite systems out of individual hilbert spaces. Thus we can deal with quantum systems having multiple components like say 3 qubits simultaneously.

In this section we will deal with making composite quantum state spaces out of elementary state spaces. This will be important for instance, when we deal with multiple qubits. We begin with tensor products which serve as the mathematical backbone behind composite systems.

3.2.1 Tensor Products

Suppose $\mathbf{A}$ and $\mathbf{B}$ are two vector spaces that are n and m dimensional. The tensor product of $\mathbf{A}$ and $\mathbf{B}$ is denoted as

$$\mathbf{A} \otimes \mathbf{B}$$

This is a vector space that is nm dimensional whose basis elements are the tensor products of the basis elements of the individual vector spaces. An informal way to think of a tensor product is that it is just a placeholder where the individual vector spaces are kept along different dimensions. Suppose $|v\rangle \in \mathbf{A}$ and $|w\rangle \in \mathbf{B}$, then

$$|v\rangle \otimes |w\rangle \in \mathbf{A} \otimes \mathbf{B}$$

.

The tensor product is a vector space and along with that it must satisfy the following properties:

1. For scalar $c \in \mathbb{C}$ and $|v\rangle \in \mathbf{A}, |w\rangle \in \mathbf{B}$

$$c\left(|v\rangle \otimes |w\rangle\right) = c\,|v\rangle \otimes |w\rangle = |v\rangle \otimes c\,|w\rangle$$

2. For $|v_1\rangle, |v_2\rangle \in \mathbf{A}, |w\rangle \in \mathbf{B}$

$$\left(|v_1\rangle + |v_2\rangle\right) \otimes |w\rangle = |v_1\rangle \otimes |w\rangle + |v_2\rangle \otimes |w\rangle$$

3. For $|v\rangle \in \mathbf{A}, |w_1\rangle, |w_2\rangle \in \mathbf{B}$

$$|v\rangle \otimes \left(|w_1\rangle + |w_2\rangle\right) = |v\rangle \otimes |w_1\rangle + |v\rangle \otimes |w_2\rangle$$

Similarly we can talk about operators on this tensor space. If M is a linear operator from $\mathbf{A} \to \mathbf{A}^{'}$ and N is a linear operator from $\mathbf{B} \to \mathbf{B}^{'}$ then $M \otimes N$ is a linear operator from $\mathbf{A} \otimes \mathbf{B} \to \mathbf{A}^{'} \otimes \mathbf{B}^{'}$ defined by

$$(M \otimes N) |i\rangle \otimes |j\rangle = M |i\rangle \otimes N |j\rangle$$

for $|i\rangle \otimes |j\rangle \in \mathbf{A} \otimes \mathbf{B}$. It is trivial to show using the above properties that the above operator is indeed linear. This is left as an exercise.

We can also define an inner product on this vector space. Let $\mathbf{A}$ be an inner product space and $\mathbf{B}$ be an inner product space. Then we define an inner product on $\mathbf{A} \otimes \mathbf{B}$ by

As an example suppose we have two qubits, then this composite system will have the bases $|0\rangle \otimes |0\rangle$, $|0\rangle \otimes |1\rangle$, $|1\rangle \otimes |0\rangle$ and $|1\rangle \otimes |1\rangle$. For convenience we can omit the middle symbol and write $|0\rangle \otimes |0\rangle$ as $|00\rangle$.

There is a useful way of writing the matrix representations of the operator formed by the tensor product of two operators using Kronecker Products. Suppose A is a linear operator whose matrix representation is given by A_{ij} where $1 \leq i \leq n$ and $1 \leq j \leq m$. Similarly consider the matrix representation of another operator B that was p rows and q columns. It can be shown that the matrix form of $A \otimes B$ is given by:

$$A \otimes B = \begin{bmatrix} A_{11}B & A_{12}B & \cdots & A_{1m}B \\ A_{21}B & A_{22}B & \cdots & A_{2m}B \\ \vdots & \vdots & \vdots & \vdots \\ A_{n1}B & A_{n2}B & \cdots & A_{nm}B \end{bmatrix}$$

where $A_{ij}B$ is a submatrix of the dimension of B formed by scaling up B with A_{ij}.

We end by mentioning that the notation $|\psi\rangle^{\otimes k}$ is written in shorthand for $|\psi\rangle \otimes |\psi\rangle \cdots \otimes |\psi\rangle$ with the tensor product being done k times.

3.2.2 Mixed Quantum States

We will now study quantum states that are mixed. Formally suppose you have a quantum system that can be in one of the states from the set $\{|\psi_i\rangle\}$ with probability $\{p_i\}$. Such a system is said to be in a mixed quantum state. It is important to understand that this is not the same as a superposition. Our system is in one of the states, we simply do not know which and it has a probability corresponding to that state. We then define the density operator corresponding to that state as

$$\rho = \sum_i p_i |\psi\rangle \langle\psi|$$

The above is useful because we can rewrite all of quantum mechanics in the form of density matrices only. For instance suppose we allow a unitary transformation U to take place on our system. We can show that the final density matrix will be given by

$$\rho^{'} = U\rho U^{'}$$

Suppose an ensemble of measurement operators $\{M_m\}$ are to used to measure an observable on the mixed state ρ. We can show that the probability of obtaining a result m is given by

$$p(m) = \mathrm{tr}\big(M_m^\dagger M_m \rho\big)$$

After we obtain a measurement result m it can be shown using conditional probability that the final mixed state of the system is,

$$\rho^{'} = \frac{M_m \rho M_m^\dagger}{\mathrm{tr}\Big(M_m^\dagger M_m \rho\Big)}$$

We must note that if a system has only state in this ensemble (that is we are sure of which state it is in) then it is stated to be in a pure state. The density matrix corresponding to this state $|\psi\rangle$ is then simply

$$\rho = |\psi\rangle \langle\psi|$$

Density matrices for composite systems can be found in a similar manner by taking the tensor products of the individual density matrices. We also must note that given an ensemble of states there exists a unique density matrix but the reverse is not true. Given a density matrix there may be many ensembles that correspond to this density matrix. You can try to generate such a pair of states with the same density matrix for a simple density matrix. Below we state without proof the relation between two set of ensembles such that they have the same density matrix.

Theorem 3.2.1. Two set of ensembles $\{|\psi\rangle\}$ and $\{|\phi\rangle\}$ have the same density matrix if there exists a unitary matrix of complex numbers U such that

$$|\psi_i\rangle = \sum_j u_{ij} |\phi_j\rangle$$

where add extra 0s to the smaller ensemble so that both the ensembles have the same number of elements.

It is of interest to us to find the density matrix corresponding to a subsystem of a quantum system. This is done using partial traces and we will go over it in brief.

3.2.3 Reduced Density Matrices

Suppose the composite system composed of two components A, B has a density matrix given by p_{AB}. The density matrix of a component A is given by

$$\rho_A = \mathrm{tr}_B(\rho_{AB})$$

where tr_B is the partial trace, where the trace operator is applied only on the second component. For instance when we expand the sum in ρ_{AB} we get terms of the form

$$|\psi_a\rangle \langle\psi_a| \otimes |\psi_b\rangle \langle\psi_b|$$

The partial trace of this will be

$$\mathrm{tr}_B(|\psi_a\rangle \langle\psi_a| \otimes |\psi_b\rangle \langle\psi_b|) = |\psi_a\rangle \langle\psi_a| \,\mathrm{tr}(|\psi_b\rangle \langle\psi_b|) = |\psi_a\rangle \langle\psi_a|$$

We can see why this works as suppose the individual systems had density matrices ρ_A and ρ_B then

$$\rho_{AB} = \rho_A \otimes \rho_B$$

which means that

$$\mathrm{tr}_B(\rho_{AB}) = \rho_A \,\mathrm{tr}(\rho_B) = \rho_A$$

which has thus extracted the individual density matrix out.

It is important to note that the density matrix alone is sufficient to determine not only the measurement statistics but also the state of the system with time. Density matrices thus provide an alternative to dealing with state vectors in quantum mechanics.

Chapter 4

Introduction to Computer Science and Classical Ideas

4.1 Introduction

Our main motivation behind using quantum computation is to provide an improvement over the classical model of computation. So to learn quantum phenomena, it is imperative to have a basic idea about the classical constructs and techniques that are already available. We also need to have a proper notion of how we compare two different models of computation. Many problems that are difficult to solve using classical methods can possibly have quantum algorithms that solve these problems with ease. For example, the factorisation problem of finding the prime factors of a number N, takes exponentially more time classically than via the quantum Shor's algorithm. In this chapter, I will start with the Turing model and circuit model for computation and establish the relationship between these two models. Then I will move onto how we can compare algorithms and finally end this section by describing different classes of algorithms on the basis of their complexities.

4.2 Turing Machines

The field of computer science arose back in the 1930s when the pioneers of the field Alan Turing and Alonzo Church formalized the notion of an algorithm used to solve mathematical problems by providing a model of computation. Turing's model, called the Turing Machine provides an abstract notion of a machine, that in principle can solve all problems associated with an algorithmic process. We will discuss its construction now.

A Turing machine contains four main elements:

1. a program, rather like an ordinary computer which contains the instructions corresponding to the algorithm
2. a finite state control, which acts like a stripped-down microprocessor working with the other components and containing the states of the machine
3. a tape which is the memory/output of the machine
4. a read write tape head, which points to the current position on the tape which is being accessed.

The finite state control for a Turing machine consists of a finite set of internal states, $q_1, q_2 \cdots q_m$. The number m can be varied; it turns out that for m sufficiently large this does not affect the power of the machine in any essential way, so without loss of generality we may suppose that m is some fixed constant. Along with these m states, there are two more states, the starting state q_s and the halting state q_h which

denote the start and the end of the process respectively. This means that when the processing starts, the machine will be in the starting state, and when the process is completed, the state control reaches the halting state. These states can be thought of as the internal storage of the machine which help in the processing.

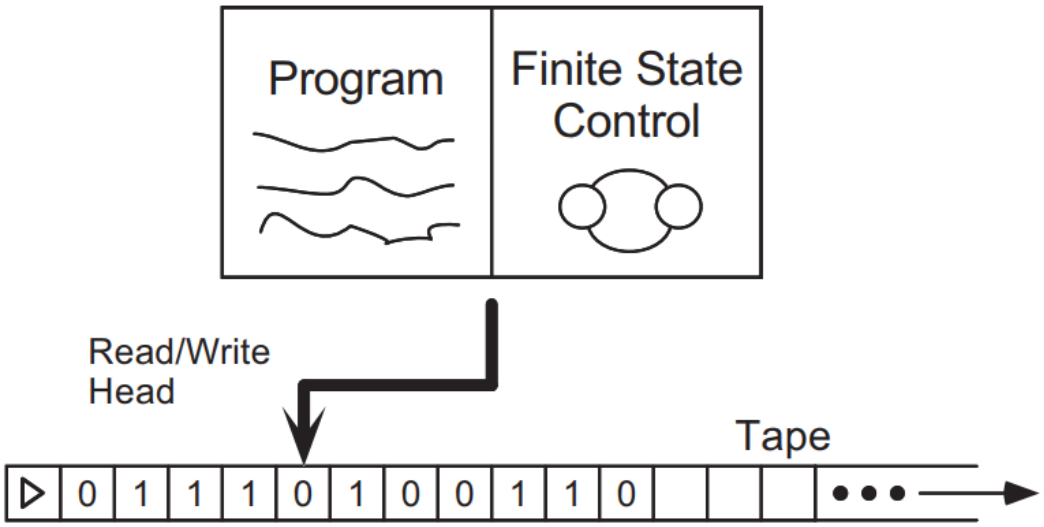
The Turing machine tape is a one-dimensional object, which stretches off to infinity in one direction. The tape can be thought of as an infinite sequence of squares. The tape squares each contain one symbol drawn from some *alphabet*, Γ, which contains a finite number of distinct symbols. For simplicity, let's assume the alphabet consists of only 4 symbols, which we denote by 0, 1, b ('the blank symbol'), and $\triangleright$, to mark the left hand edge of the tape. The tape will start with $\triangleright$ and then contain some zeros and ones and finally have blanks on it. The *read-write tape-head* identifies the current square on the Turing machine tape and is responsible for writing the new value on the square as the machine moves onto the next square. This can be regarded as both the input and the output of the device.

Let us see how the turing machine works. I will illustrate this via an example which will show an explicit construction of a turing machine that outputs the function $f(x) = 1$. Firstly let us see how the program of the machine is defined. Formally a *program* in a turing machine is a list of tuples (which correspond to commands) of the form

$$\left(q_i, x, q_j, x^{'}, d\right)$$

where q_i is the current state in which the machine is supposed to be, x is the value on the current square in the tape head, q_j is the state the machine goes into after executing the current command, x' is the value that the machine will write on the current square after this command is executed and d is either $0, +1, -1$ where a value of 0 instructs the machine to not move the tape head, $+1$ means move the pointer to the right and -1 means move the tape to the left. The only exception to this rule is when the current value on the tape is $\triangleright$ which would ignore the -1 instruction as we are at the beginning of the tape.
The machine will obtain its current working condition by reading the value on the tape square and scan for the program line that has its current state and this value as the first two elements of the tuple. If such a line does not exist it automatically goes to q_h and halts execution.

Figure 4.1: A Turing Machine



Now we will list an explicit program for computing $f(x) = 1$. Consider the following program:

1. $(q_s, \triangleright, q_1, \triangleright, +1)$
2. $(q_1, 0, q_1, b, +1)$
3. $(q_1, 1, q_1, b, +1)$

4. $(q_1, b, q_2, b, -1)$
5. $(q_2, b, q_2, b, -1)$
6. $(q_2, \triangleright, q_3, \triangleright, +1)$
7. $(q_3, b, q_h, 1, 0)$

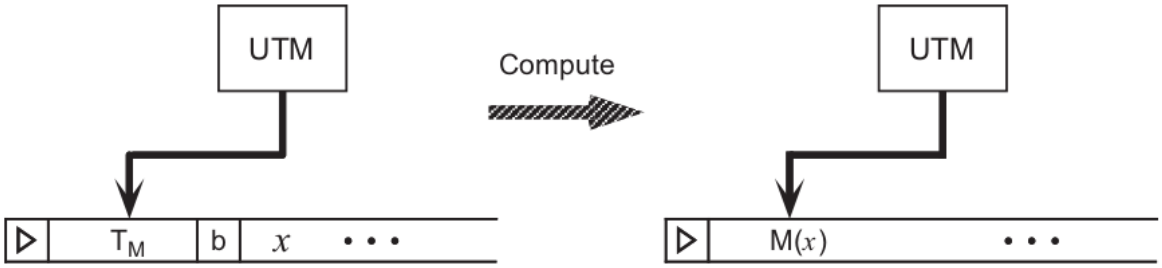
We can see that this program outputs $f(x) = 1$ followed by a series of blanks. You can verify this by tracing the working of the machine line by line.

Let us see some variations of turing machines. For instance, instead of having just one single tape, the machine could contain two tapes wherein you can use one for the input and other for the output. This variation is identical to the simple turing machine in the sense that both can compute the same set of functions. However for explicitly writing programs for a turing machine it is indeed more convenient to work with the two tape version. A program line of this turing machine will be of the form $\left(q, x_1, x_2, q^{'}, x_1^{'}, x_2^{'}, d_1, d_2\right)$ where we will use subscript 1 for the first tape and 2 for the other tape.

4.2.1 The Universal Turing Machine

We've described Turing machines as containing three elements which may vary from machine to machine – the initial configuration of the tape, the internal states of the finite state control, and the program for the machine. A clever idea known as the *Universal Turing Machine*(UTM) allows us to fix the program and finite state control once and for all, leaving the initial contents of the tape as the only part of the machine which need to be varied. The Universal Turing Machine has the following property. Let M be any Turing machine, and let T_M be the Turing number associated to machine M . Then, on input of the binary representation for T_M followed by a blank, followed by any string of symbols x on the remainder of the tape, the Universal Turing Machine gives as output whatever machine M would have on input of x. Thus, the Universal Turing Machine is capable of simulating any other Turing machine!

Figure 4.2: The Universal Turing Machine



4.2.2 Church Turing Thesis

Although it seems that we certainly did not require such a complex setup to simply compute $f(x) = 1$ the beauty of turing machines lie in the fact that this process can be used to compute various functions ranging from addition, polynomial evaluation to complex arithmetic. Surprisingly, it turns out that a Turing machine can be used to simulate all the operations performed on a modern computer!. The efficiencies may be different but we are more interested in knowing which problems can indeed be solved on a turing machine. Church and Turing came up with their thesis, the *Church-Turing Thesis* that answers exactly this question.

Thesis. CHURCH TURING THESIS: The class of functions computable by a Turing machine corresponds exactly to the class of functions which we would naturally regard as being computable by an algorithm.

It is to be noted that this is just a hypothesis. Church, Turing and many other people have spent a great deal of time gathering evidence for the Church-Turing thesis, and in 60 years no evidence to the contrary has been found.

Nevertheless, it is possible that in the future we will discover in Nature a process which computes a function not computable on a Turing machine. It would be wonderful if that ever happened, because we could then harness that process to help us perform new computations which could not be performed before. Of course, we would also need to overhaul the definition of computability, and with it, computer science.

It is interesting to note that quantum computers, and in general, the quantum model of computation do not change the class of functions computable by the model, it merely provides a more efficient way to do so.

If we allow randomness in the model, we get a probabilistic Turing machine. This machine will transition from one state to another by randomly choosing a state from various possibilities. Note that, a deterministic turing machine can in essence simulate a probabilistic machine by going through all possibilities (searching throughthe search space). Thus they both can compute the exact same class of functions. And therefore we have the *Strong Church-Turing Thesis* :

Thesis. STRONG CHURCH TURING THESIS: Any model of computation can be simulated on a probabilistic Turing machine with at most a polynomial increase in the number of elementary operations required.

The meaning of polynomial increase is defined later in the computational complexity section.

Now I will describe the circuit model of computation in brief.

4.3 Circuit Model of Computation

The circuit model of computation is essentially the model we are most familiar with, wires and gates that compute binary functions. Taking multiple inputs we can actually compute all kinds of functions which are equivalent to those that can be simulated on a turing machine. In general our circuits will have gates, which are blocks that compute functions of the form $f : \{0,1\}^k \to \{0,1\}^l$. However we would like to reduce all our circuits to those made up of some universal blocks. There are a few gates called the universal gates in classical computation that can build up any gate. One such gate is the NAND gate which takes two binary inputs and computes their binary AND and then negates the result.

There are other universal gates as well and some common gates like OR, AND, XOR which are self explanatory. Two other gates are the FANOUT and crossover gates. The FANOUT gate takes an input and copies it out. We will later see that the FANOUT gate has no quantum equivalent because of the *no-cloning* theorem. The crossover gate takes two inputs and simply swaps them.

Later on when we get into quantum circuits, we will show quantum equivalents of these gates and also look at universal quantum gates. It is important to note that quantum mechanics is reversible, infact all of quantum mechanics can be represented as unitary transformations which we will look into in the next section. As a result it is not possible to have irreversible gates in our quantum computational model. So a quantum variant of an AND gate is not possible as AND is irreversible which means that given the output of the AND gate we cannot determine what the inputs were. We reserve this discussion for later when we start dealing with quantum circuits.

4.4 Analysis of Computational Problems

So far we have said that quantum computers provide a more efficient solution of various classical problems. How do we deem which algorithm is better? This section deals with that.

First I'll talk about orders of magnitude of functions. For instance, take an algorithm which scans a list of n numbers and outputs the maximum among them. Clearly such an algorithm takes n operations to finish. In general lesser the number of steps the better the algorithm. But we cannot really talk about the exact number of steps and instead we need an estimate on the number of steps as this value can be variable.

Suppose we have an algorithm that scans a list of n numbers to check if 1 is present or not and if it finds 1 it terminates. Clearly this can take any number of steps ranging from 1 to n. What we know is that at worst it takes n steps (the upper bound) and in the best case it takes 1 step (lower bound). So the ideas of upper bound and lower bound on the number of steps provide a rough estimate of how good the algorithm is, while average case analysis provide a better picture by accounting for all cases. To talk about such cases in a more formalised manner we use the big-O, big-Omega and big-theta notation.

Definition 4.4.1. Big O A function $f(n)$ is said to be $O(g(n))$ if there exists some constant c and some n_0 such that for all $n \geq n_0$,

$$f(n) \leq cg(n)$$

An algorithm A is said to have (upper) time complexity $O(g(n))$ if the function $\mathbf{TIME}(A(n)) = f(n)$ is $O(g(n))$ where $f(n)$ is the number of steps the algorithm takes on that input. For example the previous algorithm is $O(n)$ as for $c = 1$ and $n_0 = 1$ $f(n) \leq n$ is satisfied where n is the size of our input.

Similarly we have big theta and big O notations that deal with different bounds.

Definition 4.4.2. Big Theta A function $f(n)$ is said to be $\Theta(g(n))$ if there exists some constant c_1, c_2 and some n_0 such that for all $n \geq n_0$,

$$c_1 g(n) \leq f(n) \leq c_2 g(n)$$

Definition 4.4.3. Big Omega A function $f(n)$ is said to be $\Omega(g(n))$ if there exists some constant c and some n_0 such that for all $n \geq n_0$,

$$cg(n) \leq f(n)$$

So for instance, the previous algorithm is $\Omega(1)$ and $O(n)$. It is upto you to prove that we cannot have any Θ bound for this algorithm. We can also have space complexity where instead of talking about the number of steps, we refer to the number of memory units the program requires. Therefore the previous algorithm has $\Theta(n)$ space complexity as it is using roughly n memory units to store all values (measured as multiples of memory units used to store one number).

Now let us compare the algorithms for prime factorisation. If a number N is given the size of its input is $\log_2 N$. Remember that computer scientists always deal with logarithms to the base 2 as values are stored in binary. So the implicit base is 2. Now Shor's algorithm is $O(\log^3 N)$ which is a massive improvement over classical algorithms. A simple trial algorithm is $O(N)$ which becomes exponential in the input size $\log_2 N$. Notice that we are only concerned with the behaviour of the function at large values of the input as computation becomes infeasible only at those limits.

4.5 Complexity Classes

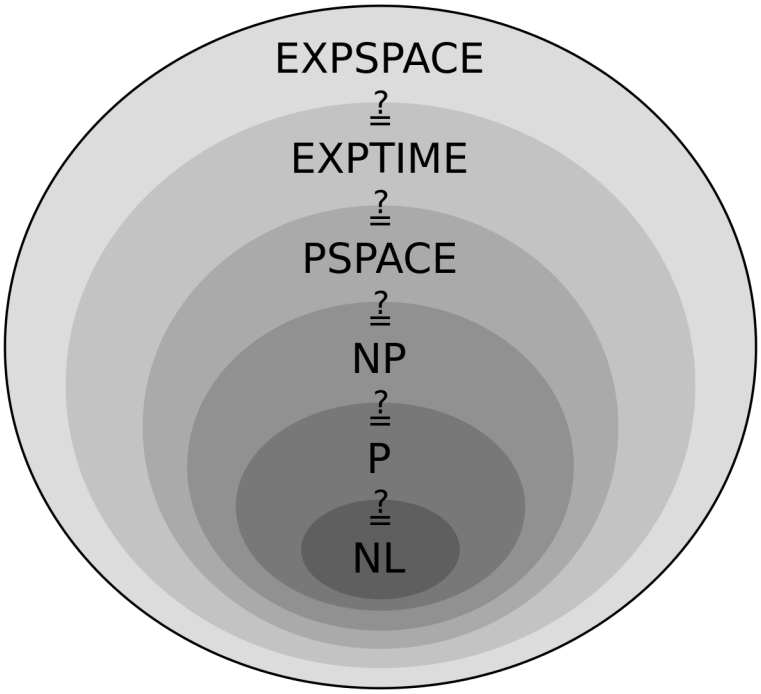
When we covered big-o notation to compare running times of different algorithms we found a way of quantifying the upper bound of the running time. Naturally it makes sense to now categorize different algorithms into different classes on the basis of their running time.

We will start by classifying decision problems, that is problems that have a yes/no answer. These problems provide a lot of insight into the generic case of computable problems and so we stick to them for this report. For instance we may ask the question whether a number n is a prime or not. This is a decision problem as it has a pure yes/no answer.

To formalize this notion of a decision problem, we rephrase it in terms of a *language* L. A language is a subset of all possible strings formed from a subset of characters which we term as alphabets. For instance we may define the *prime* language L_P as the subset of all possible strings formed from $\{0, 1\}$ such that the corresponding binary number is a prime.

Then we can rephrase decision problems into creating a Turing Machine (or an algorithm) that will output yes if the corresponding input is part of that language and else no. These outputs can be two end states of the turing machine q_y, q_n.

Figure 4.3: The various complexity classes



A complexity class is a set of problems that satisfy a particular property with respect to its time or space or energy complexity. For an algorithm (or a machine implementing it) A solving a decision problem D we say that $D \in \mathbf{P}$ where $\mathbf{P}$ is a complexity class if for an input of size n, $\mathbf{TIME}(\mathrm{A(n)}) = O(n^k)$ for some $k > 0$. Roughly $\mathbf{P}$ is the collection of those decision problems where the decision can be found out in polynomial size of the input.

$\mathbf{P}$ is an important complexity class as it contains those problems that we can solve in a reasonable amount of time compared to say exponential or factorial growth algorithms.

Another important complexity class is **NP**. Roughly speaking it is that collection of decision problems whose truth value can be verified with the help of another input called the witness W in polynomial size of the main input. Formally a decision problem is in **NP** if

1. given x $\in$ L, there exists a witness W such that using the W a turing machine can deterministically state that x $\in$ L by going to the positive state q_y in time proportional to a polynomial in size of x.
2. given x $\notin$ L, for all possible witnesses W the turing machine will deterministically state that x $\notin$ L by going to the state q_n in time proportional to a polynomial in size of x.

For example consider the decision problem D asking whether a number x is **NOT** in the prime language, that is $x \in L_P^{'}$ that we previously described. Then a possible witness could be a factor of the number x where we check if the witness divides the number (where the witness is not x or 1). If x was a prime then for any possible witness we cannot use it to say that witness divides the number and so the second condition is met. Thus this problem is in **NP**.

In general it is not easy to provide a witness for the complement of a language. For example in the previous case the complement would be the language of primes. There, coming up with a witness that gives a guarantee that $x \in L_P$ is not easy.

Chapter 5

Quantum gates and circuits

5.1 Single Qubit Gates

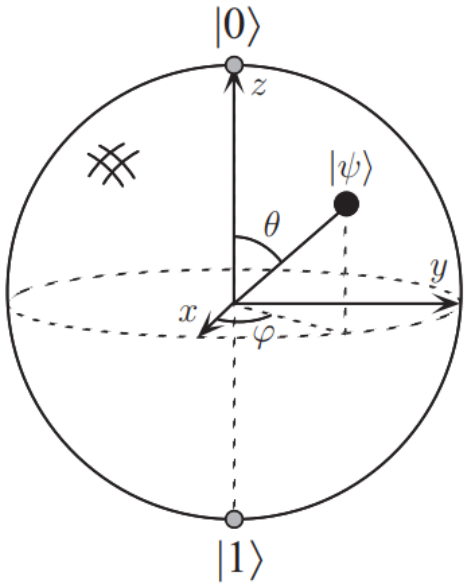
In this section we will deal with single qubit gates and different ways we can represent them.

A single qubit gate can be represented as an unitary operator acting on the state vector of the qubit.That is for every gate G there exists a unitary transformation U such that the final state of a state vector $|\psi\rangle$ is given by

$$|\psi_2\rangle = U\,|\psi\rangle$$

This is because the gate can be thought of as an abstract machine that takes in a state vector and operates on it with time. As all time transformations of quantum state vectors are unitary transformations the above result holds. Conversely we assume that for every unitary operator on a single qubit there exists a gate for the same. Achieving the construction of such a gate would venture into physical realisation of quantum computation which we reserve for later.

Figure 5.1: Bloch Representation of a Qubit



We now introduce the bloch sphere notation for qubits. Observe that we can write any qubit as follows:

$$|\psi\rangle = e^{ix}\left(\cos\frac{\theta}{2} + ie^{i\phi}\sin\frac{\theta}{2}\right)$$

where

$$0 \leq x < 2\pi, 0 \leq \theta \leq \pi, 0 \leq \phi < 2\pi$$

The global phase e^{ix} is irrelevant as it is common to both. Observe that by neglecting this our state vector is parameterised by θ and ϕ and these values are precisely in the range of the spherical angular coordinates of a sphere of radius 1. Thus corresponding to every $|\psi\rangle$ (θ, ϕ) there exists a point on the unit sphere given by $(1, \theta, \phi)$ in spherical polar coordinates. This sphere is termed as the bloch sphere and the corresponding position vector the bloch vector.

It is interesting to note that any unitary transformation in the state space can be visualised to be a rotation of the bloch sphere about an axis. That is this mapping corresponds to a rotation of the bloch sphere (upto a global phase). You will prove this result later on. We first define some standard gates in their matrix forms. These gates are incredibly useful and are known as the pauli matrices.

First we would like to describe some standard transformations (or gates) that rotate the sphere about x, y, z axes.

$$R_x(\theta) = e^{-i\theta\frac{X}{2}} = \begin{bmatrix} \cos\frac{\theta}{2} & -i\sin\frac{\theta}{2} \\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{bmatrix}$$

$$R_y(\theta) = e^{-i\theta\frac{Y}{2}} = \begin{bmatrix} \cos\frac{\theta}{2} & -\sin\frac{\theta}{2} \\ \sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{bmatrix}$$

$$R_z(\theta) = e^{-i\theta\frac{Z}{2}} = \begin{bmatrix} e^{\frac{-i\theta}{2}} & 0 \\ 0 & e^{i\frac{\theta}{2}} \end{bmatrix}$$

It is important to note that in the bloch sphere representation the coefficient along $|0\rangle$ is always a real positive number. After we apply any gate on the state vector this may not be true and so we must first divide by a common phase factor to ensure the above condition is met before we figure out the bloch vector.

We prove this important theorem that relates unitary transformations to rotation transformations using the rotation matrices.

Theorem 5.1.1. Suppose U is a unitary transformation. Then there exists reals α, β, γ and δ such that

$$U = e^{i\alpha}R_z(\beta)R_y(\gamma)R_z(\delta)$$

Proof. Since U is unitary, the rows and columns of U are orthonormal, from which it follows that there exist real numbers α, β, γ and δ such that

$$U = \begin{bmatrix} e^{i(\alpha-\frac{\beta}{2}-\frac{\delta}{2})}\cos\frac{\delta}{2} & -e^{i(\alpha-\frac{\beta}{2}+\frac{\delta}{2})}\sin\frac{\delta}{2} \\ e^{i(\alpha+\frac{\beta}{2}-\frac{\delta}{2})}\sin\frac{\delta}{2} & e^{i(\alpha+\frac{\beta}{2}+\frac{\delta}{2})}\cos\frac{\delta}{2} \end{bmatrix}$$

By expanding the right hand side of the statement to prove we immediately get the above and thus the given statement is proved. □

The above theorem allows us to construct any gate using just three parameterised gates. There exists an important corollary of the above that we leave as an exercise. This corollary is useful when we want to construct multi qubit gates.

We end this section by mentioning two other important gates that we will later use. The Hadamard gate was previously introduced and is given by

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

The T gate (also known as the $\frac{\pi}{8}$ gate) is given by

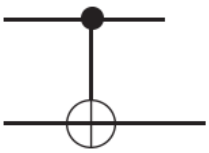
$$T = \exp(i\pi/8) \begin{bmatrix} \exp(-i\pi/8) & 0 \\ 0 & \exp(i\pi/8) \end{bmatrix}$$

5.2 Multi Qubit Gates

When we move from operations on a single qubit to operations that take in multiple qubits as inputs we gain much more flexibility. One particular type of operation is of great interest to us, a controlled operation. Classical computation is full of conditionals of the form "If A then do B" and we would like to have something similar for quantum circuits as well. Let's start with the simplest of them all, Controlled NOT gates.

5.2.1 The Controlled NOT Gate

The controlled NOT gate takes in two qubits. One is termed as the control qubit and the other is termed as the target qubit. The control qubit is responsible for controlling the operation on the target qubit and operation that we seek to achieve is the NOT operation. In a circuit CNOT is represented in the following manner:



The upper qubit is the control qubit and the lower one is the target. If the upper qubit is in $|1\rangle$ state then the NOT gate is applied on the lower qubit. Thus a state $|10\rangle$ on being sent to through this gate will turn into $|11\rangle$ while a state $|01\rangle$ will remain $|01\rangle$.

5.2.2 Universality of CNOT

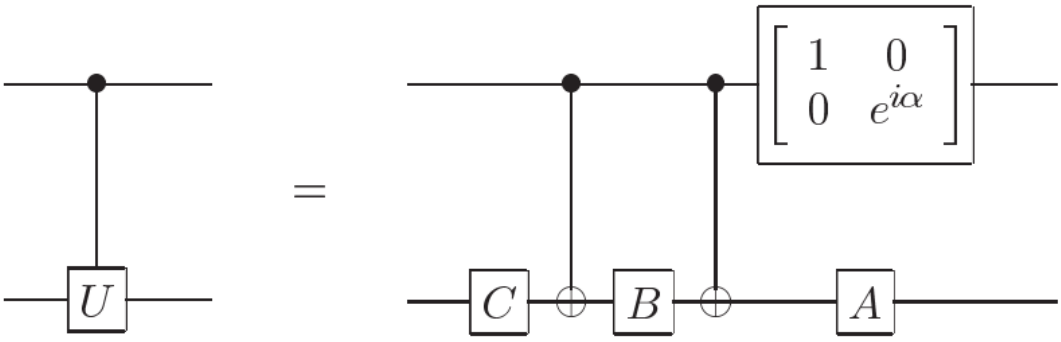
This simple gate is extremely powerful as using just a CNOT gate and some single qubit gates we can implement any other multi qubit gate. In a sense the CNOT gate is a universal gate which we need in order to construct gates that perform unitary operations on multiple qubits.

As a simple example showing how this construction can be applied in practice let us see how a controlled-U gate can be implemented using just single qubit gates and a CNOT gate. Single qubit gates on their own can be implemented using the universal single qubit gates and so we will assume that they can be implemented.

Any single qubit gate can be written in the form $U = e^{i\alpha} AXBXC$ where $ABC = I$ with A, B, C being three single qubit gates. Consider the following construction. It is left as a trivial exercise to verify that the right hand circuit is indeed implementing the left hand controlled U operation.

We can also have gates that have more than a single qubit as their control qubit. A controlled U gate with n controls is represented as $C^n(U)$

A lot can be said about multi qubit gates but essentially the main point is that it can be proven that any unitary operation on multiple qubits can be achieved upto arbitrary accuracy using hadamard, phase,



CNOT, and $\pi/8$ gates. The proof is a bit involved and we skip it but the main idea is that any unitary operation on multiple qubits can be expressed as a product of unitary operators that operate on two or less qubits. Thus if we show that we can construct any unitary operator acting on two or less qubits we are done. It is then proven that single qubit gates with CNOT gates are sufficient to implement any arbitary gate operating on two qubits. Finally we have to show that any single qubit gate can be approximated using some fixed universal gates that we mentioned above. Although it is theoretically proven implementing everything that we have discussed in practice seems quite complicated and this is one of the main challenges that physicists and computer scientists have been working on in order to bring quantum computation to fruition.

Part I

Applications and Algorithms

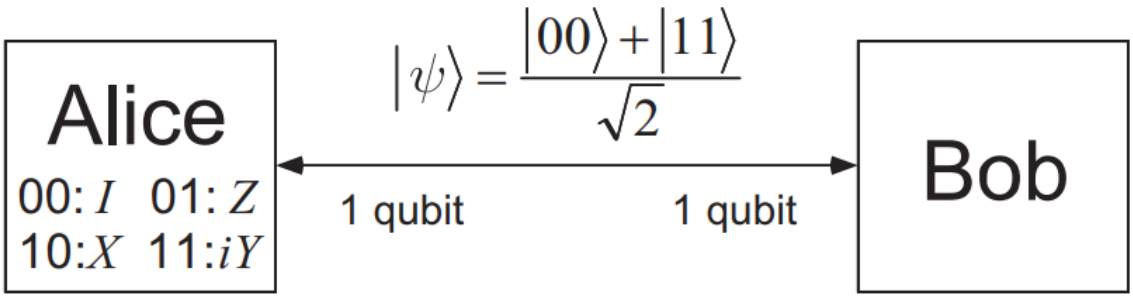
Chapter 6

Superdense Coding

Superdense coding is a simple yet surprising application of elementary quantum mechanics. It combines in a concrete, non-trivial way all the basic ideas of elementary quantum mechanics, as covered in the previous sections, and is therefore an ideal example of the information processing tasks that can be accomplished using quantum mechanics. Superdense coding involves two parties, conventionally known as 'Alice' and 'Bob', who are a long way away from one another. Their goal is to transmit some classical information from Alice to Bob. Suppose Alice is in possession of two classical bits of information which she wishes to send Bob, but is only allowed to send a single qubit to Bob. Is this possible?

Interestingly, superdense coding tells us that the answer to this question is yes.

Figure 6.1: The setup of superdense coding with Alice and Bob each possessing one qubit which are entangled



The EPR pairs or the bell states refer to the four following states comprising of two qubits that are entangled with each other.

$$\frac{|00\rangle + |11\rangle}{\sqrt{2}}, \frac{|00\rangle - |11\rangle}{\sqrt{2}}, \frac{|10\rangle + |01\rangle}{\sqrt{2}}, \frac{|01\rangle - |10\rangle}{\sqrt{2}}$$

The main idea is that we will use the entanglement of the two qubits in one of these states to transfer two bits of information by simply transferring one bit. Suppose Alice wants to send Bob two classical bits of information. To do so using superdense coding, an external party will first prepare the state

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

and he will then give the first qubit to Alice and the second to Bob. If Alice wants to send the information 00 she will do nothing and pass her qubit to Bob. If she wants to send 01 then she applies the phase flip gate Z, if she wants to send 10 she applies the quantum NOT gate X to her qubit, if she wants to send 11

she applies the iY gate. The final state of the entangled qubits corresponding to the information she wants to send is:

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

$$01 : |\psi\rangle = \frac{|00\rangle - |11\rangle}{\sqrt{2}}$$

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

$$11 : \frac{|01\rangle - |10\rangle}{\sqrt{2}}$$

Notice that all four of these states form a basis and thus when Alice passes her qubit to Bob then Bob can find out which state he got by simply measuring with respect to this basis to figure out which basis vector he had received.

Notice that this process did involve two qubits but Alice had to send only one qubit. The entanglement of the two qubits allowed Bob to figure out two classical bits of information on transfer of a single qubit.

Let's now look at the reverse, can we transmit a qubit by sending only classical information?

Chapter 7

Quantum Teleportation

This time we want Alice to send two classical bits of information to "send" a qubit. This seems a bit complicated as a qubit is determined by one of it's coefficients along a computational basis. This after taking out a global phase factor can be a real number and so may have a lot of digits in it's expansion and it seems counter intuitive that two classical bits can transmit this information.

However we again leverage the power of entanglement and the EPR pairs. We first let an external agent prepare the first bell state given by

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

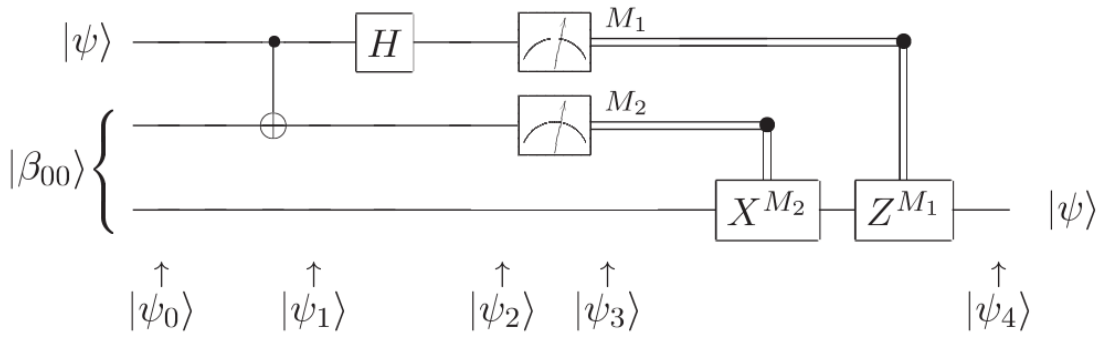
We again give the first qubit to Alice and the second to Bob. Now suppose Alice wants to send or teleport the following qubit to Bob.

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

The composite quantum state which we have considering $|\psi\rangle$ will be

$$|\psi_0\rangle = |\psi\rangle\,|\beta_{00}\rangle$$

Alice can use the following circuit to convert the composite system of the bell state and this qubit to convert the state into a form that can be used for teleportation.



Let's go through this circuit one by one. Observe that Alice only has the first two qubits. First a controlled CNOT operation with the qubit corresponding to the state we want to transfer as the control bit is applied.

This results in the state

$$|\psi_1\rangle = \frac{\alpha|0\rangle(|00\rangle + |11\rangle) + \beta|1\rangle(|10\rangle + |01\rangle)}{\sqrt{2}}$$

Then a Hadamard gate on the first qubit is applied. This results in the net state being (verify that this is indeed the third state)

$$|\psi_3\rangle = \frac{|00\rangle(\alpha|0\rangle + \beta|1\rangle) + |01\rangle(\alpha|1\rangle + \beta|0\rangle) + |10\rangle(\alpha|0\rangle - \beta|1\rangle) + |11\rangle(\alpha|1\rangle - \beta|0\rangle)}{2}$$

Now this state is rather useful for us. Observe that if we measure the first two qubits, depending upon which state the first two qubits collapse into the third qubit will be one of the following states,

$$|00\rangle : \alpha|0\rangle + \beta|1\rangle$$

$$|01\rangle : \alpha|1\rangle + \beta|0\rangle$$

$$|10\rangle : \alpha|0\rangle - \beta|1\rangle$$

$$|11\rangle : \alpha|1\rangle - \beta|0\rangle$$

Thus if Alice measures the first two qubits that she possesses, depending on her result, the third qubit with Bob will change into one of the four states. If Alice passes her measurement result information (which is given in the form of two classical bits) to Bob, Bob can fix his state to obtain the state $|\psi\rangle$ back. Verify from the diagram that applying that the operations X and Z do indeed result in the original state reverting. For instance if the result was 11 then Bob will apply both the X and Z gates while if it was 01 then only the Z gate is applied. Thus using the EPR pair we can achieve quantum teleportation.

An important point to note here is that quantum teleportation does not violate the quantum no cloning theorem. Observe that we did not copy the original state to be transferred as the original state transformed into a different state losing it's initial properties. In a sense we have transferred the information, not copied it.

Chapter 8

Deutsch-Jozsa Algorithm

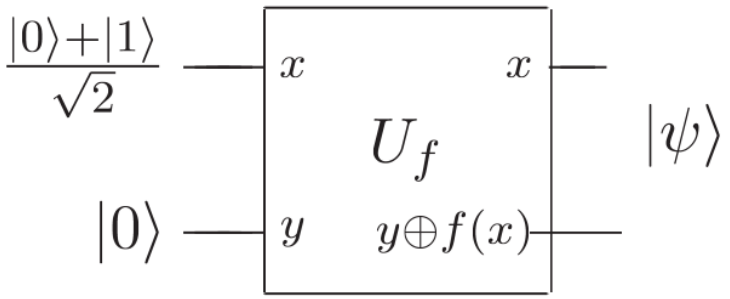
8.1 Motivation

One interesting property of quantum circuits that we want to leverage is parallelism. This follows directly from the principle of superposition. To make it a bit more clear let us consider a black box that allows us to compute a function $f : \{0,1\}^n \to \{0,1\}$. We however need a place to store the result. We thus want to have a function that achieves the following operation for us,

$$|x, y\rangle \to |x, f(x) \oplus y\rangle$$

This can be achieved with a unitary operator U_f. The process for constructing such an oracle would be to take the classical circuit for computing $f(x)$ and use quantum equivalents of the gates used in this circuit. Leaving aside this implementational detail we observe that if we pass in a superposition of states in x in principle we can obtain multiple values of the function in a single pass. For instance consider the simple case of $n = 1$ and suppose that

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



Then the final state $|\psi\rangle$ would then be simply

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

This means that we have obtained two values of the function $f(x)$ by evaluating our quantum circuit just once. But unfortunately this state is a superposition. We cannot obtain both the values as measurement once collapses this state. What we therefore desire is to use this power of superposition to obtain non trivial values that would normally require us to compute the classical circuit multiple times. The Deustch Algorithm allows to find $f(0) \oplus f(1)$ by just evaluating the circuit once.

8.2 Deustch Algorithm

In order to obtain $f(0) \oplus f(1)$ we will use the oracle after applying the hadamard gate on the input qubits $|0\rangle$ and $|1\rangle$. This gives us a form that becomes useful in determining $f(0) \oplus f(1)$ as we will see.

Refer to the figure on the next page to see the circuit we will be implementing. The state $|\psi_1\rangle$ is given by

The state $|\psi_2\rangle$ after applying the oracle can be written in the following form

$$|\psi_2\rangle = \pm \left[\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right] \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right]$$

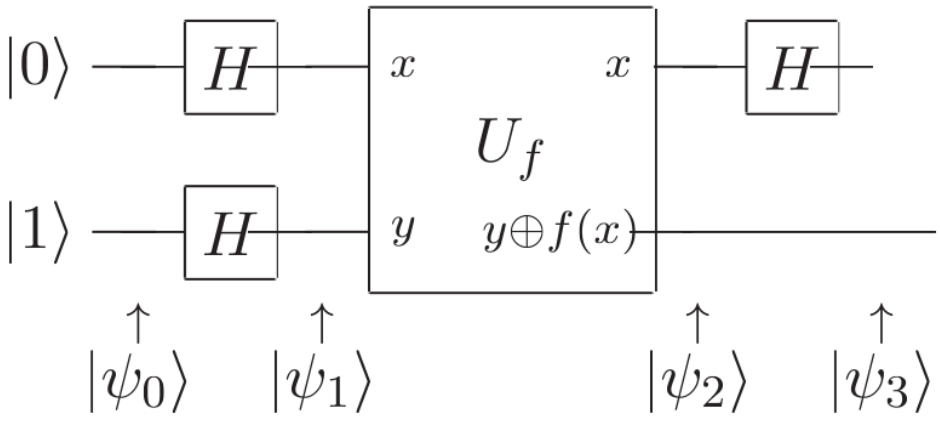
if $f(0) = f(1)$ otherwise

$$|\psi_2\rangle = \pm \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right] \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right]$$

Observe that by applying the hadamard on the first qubit and observing that $|f(0) \oplus f(1)\rangle = |0\rangle$ if $f(0) = f(1)$ and $|1\rangle$ otherwise gives the state $|\psi_3\rangle$ as

$$|\psi_3\rangle = |f(0) \oplus f(1)\rangle \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right]$$

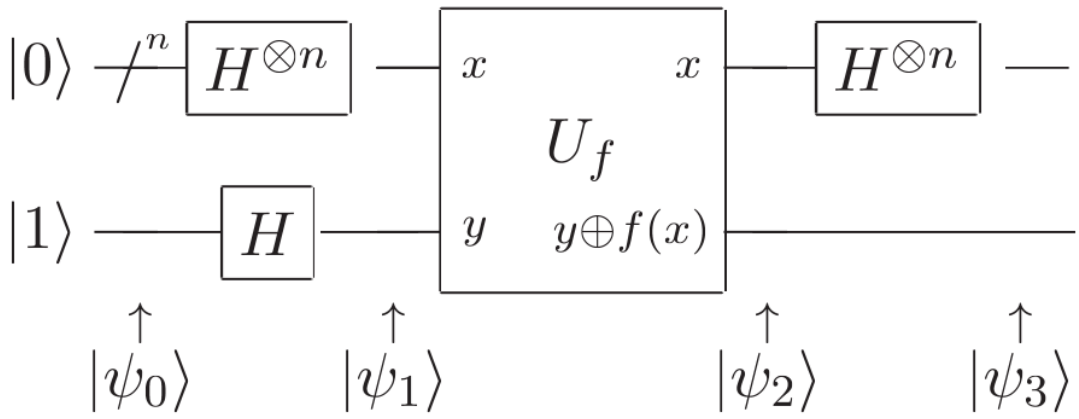
Now simply measuring the first qubit allows us to find $f(0) \oplus f(1)$ in one single evaluation of the quantum circuit. This is Deustch's algorithm.



8.3 The Deutsch-Jozsa Algorithm

Deutsch Algorithm is a special case of the more specific Deutsch-Jozsa Algorithm. Consider the following problem which can be formulated as a game.Alice, in Amsterdam, selects a number x from 0 to $2^n - 1$, and mails it in a letter to Bob, in Boston. Bob calculates some function $f(x)$ and replies with the result, which is either 0 or 1. Now, Bob has promised to use a function f which is of one of two kinds; either $f(x)$ is constant for all values of x, or else $f(x)$ is balanced, that is, equal to 1 for exactly half of all the possible x, and 0 for the other half. Alice's goal is to determine with certainty whether Bob has chosen a constant or a balanced function, corresponding with him as little as possible. Classically clearly we need atleast $2^{n-1} + 1$ classical bits to resolve this. Can we do better using quantum algorithms?

It turns out we need to evaluate the oracle only once to solve this. Notice that the previous algorithm solved exactly this for the case $n = 1$. Our modified oracle takes in n qubits and outputs the value of $f(x)$ in the target register which we set to $|1\rangle$ initially. The circuit is shown in the previous page.



Observe that we can write $|\psi_1\rangle$ as

$$|\psi_1\rangle = \frac{\sum_{x\in\{0,1\}^n} |x\rangle}{\sqrt{2^n}} \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right]$$

$|\psi_2\rangle$ then becomes

$$|\psi_2\rangle = \frac{\sum_{x\in\{0,1\}^n} (-1)^{f(x)} |x\rangle}{\sqrt{2^n}} \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right]$$

Finally we apply the n gate hadamard on the first register of n qubits. These qubits are already in superposition so it seems a bit complex to apply it on all of them. We can however write it as a double sum.

Exercise 8.3.1. Prove that

$$H^{\otimes n} |x_1, x_2 \cdots x_n\rangle = \frac{\sum_{z_1,z_2,\cdots z_n} (-1)^{x_1 z_1 + x_2 z_2 \cdots x_n z_n} |z_1, z_2 \cdots z_n\rangle}{\sqrt{2^n}}$$

The above can be written simply in the form

$$H^{\otimes n} |x\rangle = \frac{\sum_z (-1)^{x.z} |z\rangle}{\sqrt{2^n}}$$

where $x.z$ is the bitwise product modulo 2.

Thus we can write $|\psi_3\rangle$ as

$$|\psi_3\rangle = \sum_z \sum_x \frac{(-1)^{x.z+f(x)} |z\rangle}{\sqrt{2^n}} \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right]$$

Now suppose $f(x)$ is constant. Then we can see that the coefficient of $|0\rangle^{\otimes n}$ is either $+1$ or -1 depending on $f(x)$. But this means that if we measure the first n qubits in the register we will obtain $|0\rangle^{\otimes n}$ with certainty as our state vector is of unit length and so the other terms have amplitude 0. Conversely if $f(x)$ is balanced the coefficient of $|0\rangle^{\otimes n}$ is 0. Thus is we measure the qubits in the first register and if we obtain $|0\rangle^{\otimes n}$ we see that $f(x)$ is constant and otherwise it is balanced.

Thus using only one pass through the oracle we can solve this problem. This is far better than the classical case where we need to evaluate the function at $2^{n-1}+1$ values. This sums up the Deutsch-Jozsa Algorithm.

We are now at the stage where we can discuss some more complicated quantum algorithms. We start with the Quantum Fourier Transform.

Chapter 9

Introduction to the Discrete Quantum Fourier Transform

Discrete Fourier Transform acts on a vector of complex numbers, $x_0, x_1, \ldots, x_{N-1}$. of fixed length N, and outputs another vector of complex numbers,$y_0, y_1, \ldots, y_{N-1}$ given by :

$$y_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j e^{2\pi ijk/N}$$

The quantum Fourier Transform is defined in a similar way. The QFT on an orthonormal basis of vectors $|0\rangle, |1\rangle, \ldots, |N-1\rangle$ is defined to be a linear operator with the following action on the basis state:

$$|j\rangle \to \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} |k\rangle e^{2\pi ijk/N}$$

Equivalently, the action on a arbitrary state may be written as :

$$\sum_{j=0}^{N-1} x_j |j\rangle = \sum_{k=0}^{N-1} y_k |k\rangle$$

With a little algebra the quantum Fourier transform can be given the following useful product representation:

$$|j_1, j_2 \cdots j_n\rangle \to \frac{\left(|0\rangle + e^{2\pi i 0.j_n} |1\rangle\right)\left(|0\rangle + e^{2\pi i 0.j_{n-1}j_n} |1\rangle\right) \cdots \left(|0\rangle + e^{2\pi i 0.j_1 j_2 \cdots j_n} |1\rangle\right)}{\sqrt{2^n}}$$

The circuit below computes the quantum fourier transform

where R_k is defined to be the gate acting on a single qubit given by the matrix

$$R_k = \begin{bmatrix} 1 & 0 \\ 0 & e^{2\pi i/2^k} \end{bmatrix}$$

To see why observe that the hadamard gate on a qubit $|j_1\rangle$ changes it to

$$|j_1\rangle \to \frac{\left(|0\rangle + e^{2\pi 0.j_1} |1\rangle\right)}{\sqrt{2}}$$

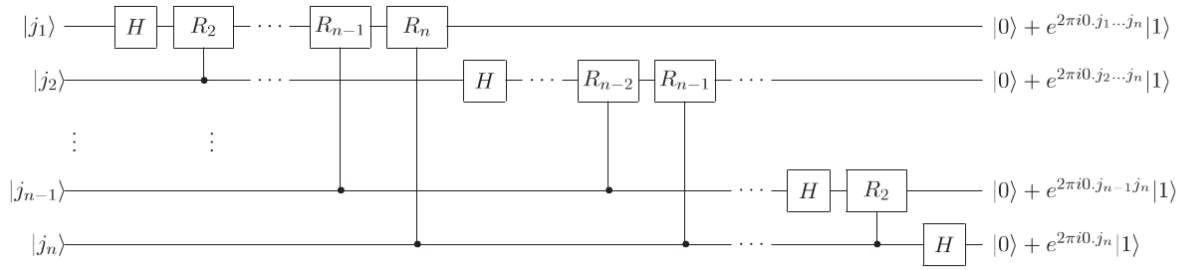


Figure 9.1: Efficient circuit for the quantum Fourier Transform

Successive applications of the controlled R gates then finally take $|j_1\rangle$ to

$$|j_1\rangle \rightarrow \frac{\left(|0\rangle + e^{2\pi 0.j_1 j_2 \cdots j_n}\,|1\rangle\right)}{\sqrt{2}}$$

We repeat the same process with the other input bits and then take their tensor product to get the form of quantum fourier transform that we obtained above. Notice that we will have to use swap gates (cross over gates that swap qubits) in order to obtain the order given in the compact form above.

How many gates does this circuit use? We start by doing a Hadamard gate and $n-1$ conditional rotations on the first qubit - a total of n gates. This is followed by a Hadamard gate and $n-2$ conditional rotations on the second qubit, for a total of $n+(n-1)$ gates. Continuing in this way, we see that $n+(n-1)+\cdots+1 = \frac{n(n+1)}{2}$ gates are required, plus the gates involved in the swaps. At most $\frac{n}{2}$ swaps are required, and each swap can be accomplished using three controlled-NOT gates. Therefore, this circuit provides a $O(n^2)$ algorithm for performing the quantum Fourier transform. In contrast, the best classical algorithms for computing the discrete Fourier transform on 2^n elements are algorithms such as the Fast Fourier Transform (FFT), which com- pute the discrete Fourier transform using $O(n2^n)$ gates. That is, it requires exponentially more operations to compute the Fourier transform on a classical computer than it does to implement the quantum Fourier transform on a quantum computer.

Can we use the quantum Fourier transform to speed up the computation of these Fourier transforms? Unfortunately, the answer is that there is no known way to do this. The problem is that the amplitudes in a quantum computer cannot be directly accessed by measurement. Thus, there is no way of determining the Fourier transformed amplitudes of the original state. Worse still, there is in general no way to efficiently prepare the original state to be Fourier transformed. Thus, finding uses for the quantum Fourier transform is more subtle than we might have hoped. We will develop several algorithms based upon a more subtle application of the quantum Fourier transform.

9.1 Phase Estimation

The Fourier transform is the key to a general procedure known as phase estimation, which in turn is the key for many quantum algorithms. Suppose a unitary operator U has an eigenvector $|u\rangle$ with eigenvalue $e^{2\pi i\varphi}$, where the value of φ is unknown. The goal of the phase estimation algorithm is to estimate φ. To perform the estimation we assume that we have available black boxes (sometimes known as oracles) capable of preparing the state $|u\rangle$ and performing the controlled-U^{2^j} operation, for suitable non-negative integers j. The use of black boxes indicates that the phase estimation procedure is not a complete quantum algorithm in its own right. Rather, you should think of phase estimation as a kind of 'subroutine' or 'module' that, when combined with other subroutines, can be used to perform interesting computational tasks. The quantum phase estimation procedure uses two registers. The first register contains t qubits initially in the state $|0\rangle$. How we choose t depends on two things: the number of digits of accuracy we wish to have in our estimate

for φ, and with what probability we wish the phase estimation procedure to be successful. The dependence of t on these quantities emerges naturally from the following analysis.

The second register begins in the state $|u\rangle$, and contains as many qubits as is necessary to store $|u\rangle$. Phase estimation is performed in two stages. First, we apply the circuit shown in Figure 8.2. The circuit begins by applying a Hadamard transform to the first register, followed by application of controlled-U operations on the second register, with U raised to successive

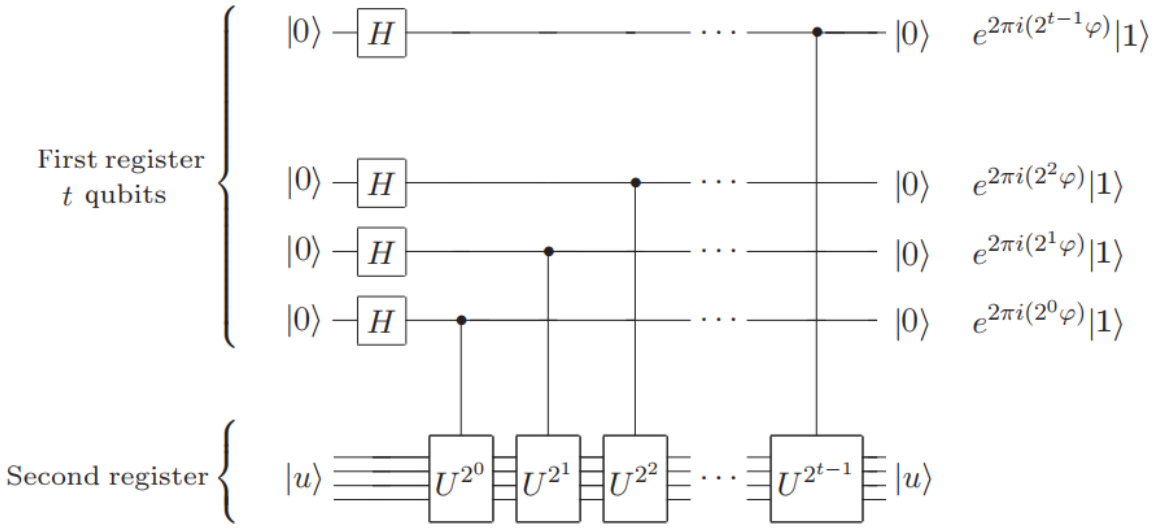


Figure 9.2: The first stage of the phase estimation procedure. Normalization factors of $\frac{1}{\sqrt{2}}$ have been omitted, on the right.

powers of two. The final state of the first register is easily seen to be

$$\frac{1}{2^{t/2}}(|0\rangle + e^{2\pi 2^{t-1}}\varphi)(|0\rangle + e^{2\pi 2^{t-2}}\varphi)\ldots(|0\rangle + e^{2\pi 2^{0}}\varphi) = \frac{1}{2^{t/2}}\sum_{k=0}^{2^t-1} e^{2\pi i\varphi k}\,|k\rangle$$

The second stage of phase estimation is to apply the inverse quantum Fourier transform on the first register. This is obtained by reversing the circuit for the quantum Fourier transform, and can be done in $O(t^2)$ steps. The third and final stage of phase estimation is to read out the state of the first register by doing a measurement in the computational basis. We will show that this provides a pretty good estimate of φ. An overall schematic of the algorithm is shown in Figure 8.3. To sharpen our intuition as to why phase estimation works, suppose φ may be expressed exactly in t bits, as $\varphi = 0.\varphi_1 ... \varphi_t$. Then the resulting from the first stage of phase estimation may be rewritten:

$$\frac{1}{2^{t/2}}(|0\rangle + e^{2\pi i 0.\varphi_t}\,|1\rangle)(|0\rangle + e^{2\pi i 0.\varphi_t\varphi_{t-1}}\,|1\rangle)\ldots(|0\rangle + e^{2\pi i 0.\varphi_1\varphi_2\ldots\varphi_t}\,|1\rangle)$$

The second stage of phase estimation is to apply the inverse quantum Fourier transform. But comparing the previous equation with the product form for the Fourier transform, we see that the output state from the second stage is the product state $|\varphi_1\varphi_2\ldots\varphi_t\rangle$. A measurement in the computational basis therefore gives us φ exactly!

Summarizing, the phase estimation algorithm allows one to estimate the phase φ of an eigenvalue of a unitary operator U, given the corresponding eigenvector $|u\rangle$. An essential feature

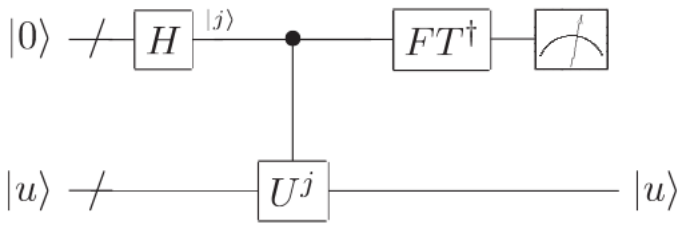


Figure 9.3: Schematic of the overall phase estimation procedure.$|u\rangle$ is an eigenstate of U witheigenvalue $e^{2\pi i\varphi}$. The output of the measurement is an approximation to φ accurate to $t - log(2 + \frac{1}{2\epsilon})$ bits, with probability of success at least 1 - ϵ

at the heart of this procedure is the ability of the inverse Fourier transform to perform the transformation:

$$\frac{1}{2^{t/2}} \sum_{j=0}^{2^t-1} e^{2\pi i\varphi} |j\rangle |u\rangle \rightarrow |\phi\rangle |u\rangle$$

, where ϕ denotes a state which is a good estimator for φ when measured.

9.2 Order Estimation

For positive integers x and N, x ¡ N, with no common factors, the order of x modulo N is defined to be the least positive integer, r, such that $x^r = 1(modN)$. The order-finding problem is to determine the order for some specified x and N. Order-finding is believed to be a hard problem on a classical computer, in the sense that no algorithm is known to solve the problem using resources polynomial in the $O(L)$ bits needed to specify the problem, where $L \equiv [log(N)]$ is the number of bits needed to specify N. In this section we explain how phase estimation may be used to obtain an efficient quantum algorithm for order-finding. The quantum algorithm for order-finding is just the phase estimation algorithm applied to the unitary operator:

$$U |y\rangle \equiv |xy(modN)\rangle$$

with y $\epsilon 0, 1^L$. (Note that here and below, when $N \leq y \leq 2^L - 1$, we use the convention that xy(mod N)is just y again. That is, U only acts non-trivially when $0 \leq y \leq N - 1$.) A simple calculation shows that the states defined by:

$$|u_s\rangle \equiv \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} exp(\frac{-2\pi isk}{r}) \left|x^k modN,\right\rangle$$

for integer $0 \leq s \leq r - 1$ are eigenstates of U, since

$$U |u_s\rangle \equiv \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} exp(\frac{-2\pi isk}{r}) \left|x^{k+1} modN\right\rangle$$

$$= exp(\frac{2\pi isk}{r}) |u_s\rangle$$

Using the phase estimation procedure allows us to obtain, with high accuracy, the corresponding eigenvalues $exp(\frac{2\pi is}{r})$, from which we can obtain the order r with a little bit more work. There are two important requirements for us to be able to use the phase estimation procedure: we must have efficient procedures to implement a controlled-U^{2^j} operation for any integer j, and we must be able to efficiently prepare an eigenstate $|u_s\rangle$ with a non trivial eigenvalue, or at least a superposition of such eigenstates.

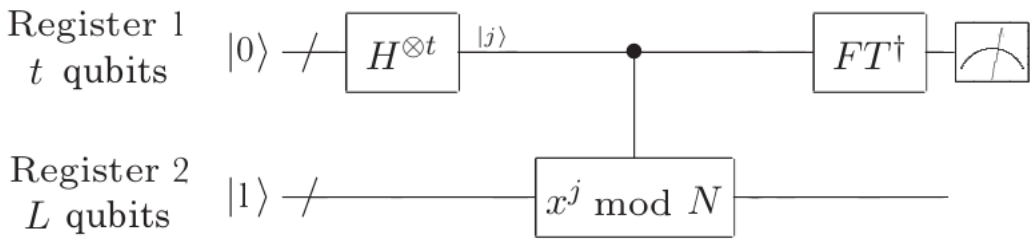


Figure 9.4: Quantum circuit for the order finding algorithm. This circuit can also be used for factoring, using a reduction we will see soon.

9.2.1 Modular exponentiation:

The first requirement is satisfied by using a procedure known as modular exponentiation, with which we can implement the entire sequence of controlled-U^{2^j} operations applied by the phase estimation procedure using $O(L^3)$ gates. We wish to compute the transformation

$$|z\rangle\, y \rightarrow |z\rangle\, U^{z_t 2^{t-1}} \ldots U^{z_1 2^0}\, |y\rangle$$

$$= |z\rangle\, |ketx^{z_t 2^{t-1}} \ldots x^{z_1 2^0} y(modN)$$

$$= |z\rangle\, |x^z y(modN)\rangle$$

Thus the sequence of controlled-U^{2^j} operations used in phase estimationis equivalent to multiplying the contents of the second register by the modular exponential $x^z(modN)$, where z is the contents of the first register. This operation may be accomplished easily using the techniques of reversible computation. The basic idea is to reversibly compute the function $x^z mod(N)$ of z in a third register, and then to reversibly multiply the contents of the second register by $x^z mod(N)$, using the trick of uncomputation to erase the contents of the third register upon completion.

The second requirement is a little trickier: preparing $|u_s\rangle$ requires that we know r, so this is out of the question. Fortunately, there is a clever observation which allows us to circumvent the problem of preparing $|u_s\rangle$, which is that

$$\frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} |u_s\rangle = |1\rangle$$

In performing the phase estimation procedure, if we use $t = 2L + 1 + [log(2 + \frac{1}{2\epsilon})$ qubits in the first register (referring to Figure 8.4), and prepare the second register in the state $|1\rangle$ - which is trivial to construct - it follows that for each s in the range 0 through r - 1, we will obtain an estimate of the phase $\varphi \approx s/r$ accurate to 2L + 1 bits, with probability at least $\frac{1-\epsilon}{r}$

9.2.2 Continued fractions algorithm

The reduction of order-finding to phase estimation is completed by describing how to obtain the desired answer, r, from the result of the phase estimation algorithm, $\varphi \approx s/r$. We only know φ to 2L + 1 bits, but we also know a priori that it is a rational number - the ratio of two bounded integers - and if we could compute the nearest such fraction to φ we might obtain r. Remarkably, there is an algorithm which accomplishes this task efficiently, known as the continued fractions algorithm. It is easily understood by example. Suppose we are trying to decompose $\frac{31}{13}$ as a continued fraction. Then it can be done by following this algorithm:

$$\frac{31}{13} = 2 + \frac{5}{13} = 2 + \frac{1}{\frac{13}{5}} = 2 + \frac{1}{2 + \frac{3}{5}} = 2 + \frac{1}{2 + \frac{1}{\frac{5}{3}}} = 2 + \frac{1}{2 + \frac{1}{1 + \frac{2}{3}}} = 2 + \frac{1}{2 + \frac{1}{1 + \frac{1}{\frac{3}{2}}}} = 2 + \frac{1}{2 + \frac{1}{1 + \frac{1}{1 + \frac{1}{2}}}}$$

9.3 Shor's factoring algorithm

Given a positive composite integer N, what prime numbers when multiplied together equal it? This factoring problem turns out to be equivalent to the order-finding problem we saw earlier, in the sense that a fast algorithm for order-finding can easily be turned into a fast algorithm for factoring. In this section we explain the method used to reduce factoring to order-finding, and give a simple example of this reduction. The reduction of factoring to order-finding proceeds in two basic steps. The first step is to show that we can compute a factor of N if we can find a non-trivial solution $x \neq \pm 1(modN)$ to the equation $x^2 = 1(modN)$. The second step is to show that a randomly chosen y co-prime to N is quite likely to have an order r which is even, and such that $y^{r/2} = \pm 1(modN)$, and thus $x \equiv y^{r/2}(modN)$ is a non-trivial solution to $x^2 = 1(modN)$. The algorithm for reduction of the factoring problem to order finding is as follows:

Inputs: A composite number N.

Outputs: A non-trivial factor of N.

Runtime: $O((logN)^3)$ operations. Succeeds with probability $O(1)$.

Procedure:

1. If N is even, return the factor 2.
2. Determine whether $N = a^b$ for integers $a \geq 1$ and $b \geq 2$, and if so return the factor a.
3. Randomly choose x in the range 1 to N - 1. If $gcd(x, N) > 1$ then return the factor gcd(x, N).
4. Use the order-finding subroutine to find the order r of x modulo N.
5. If r is even and $x^{r/2} \neq -1(modN)$ then compute $gcd(x^{r/2} - 1, N)$ and $gcd(x^{r/2} + 1, N)$,and test to see if one of these is a non-trivial factor, returning that factor if so. Otherwise, the algorithm fails.

9.4 Period-finding

Consider the following problem. Suppose f is a periodic function producing a single bit as output and such that f(x + r) = f(x), for some unknown $0 < r < 2^L$, where x, r $\epsilon 0, 1, 2, \ldots$. Given a quantum black box U which performs the unitary transform $U|x\rangle|y\rangle \rightarrow |x\rangle|y \oplus f(x)\rangle$ (where $\oplus$ denotes addition modulo 2) how many black box queries and other operations are required to determine r? Note that in practice U operates on a finite domain, whose size is determined by the desired accuracy for r. Here is a quantum algorithm which solves this problem using one query, and $O(L^2)$ other operations:

Inputs:

1. A black box which performs the operation $U|x\rangle key y = |x\rangle|y \oplus f(x)\rangle$,
2. A state to store the function evaluation, initialized to $|0\rangle$
3. $t = O(L + log(1/\epsilon))$ qubits initialized to $|0\rangle$

Output: The least integer $r > 0$ such that f(x+r) = f(x)

Runtime: One use of U, and $O(L^2)$ operations. Succeeds with probability $O(1)$.

Chapter 10

Quantum Search Algorithms

10.1 Introduction

Let us look at another classical computing task that can be sped up using the superposition principle. Our goal is to solve the needle in a haystack problem, where we have to search for an element that satisfies a particular property that lies in a haystack of N elements. Classically it seems that since this element can be any one of those N elements, it will take $O(N)$ steps to do this task. However the quantum algorithm, Grover's algorithm allows us to do this in $O(\sqrt{N})$ time. Let us closely look at this algorithm by formulating the problem first.

Suppose $f(x)$ is a function from $\{0, 1 \ldots, N-1\}$ to $\{0, 1\}$. Only for one value a, $f(a) = 1$. We seek to find this a. Also assume that $N = 2^n$. This way we can work with n qubits (if N is not of this form we can add extra elements to ensure this). Grover's algorithm involves using rotations in a 2D subspace so that this speedup ends up working.

10.2 Grover's Algorithm

Before we discuss the specifics, let us go through the main idea. Let us define two new state vectors:

$$|\psi_0\rangle = \sum_{i=0}^{N-1} \frac{|i\rangle}{\sqrt{N}}$$

$$|e\rangle = \sum_{i=0, i\neq a}^{N-1} \frac{|i\rangle}{\sqrt{N-1}}$$

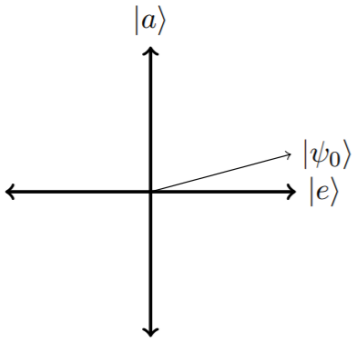
Our goal is to ensure that we obtain $|a\rangle$ in the end. Notice that

$$|\psi_0\rangle = \frac{\sqrt{N-1}\,|e\rangle + |a\rangle}{\sqrt{N}}$$

Thus $|\psi_0\rangle$ lies in the subspace spanned by $|e\rangle$ and $|a\rangle$ The coefficient of $|e\rangle$ is in general far greater than that of $|a\rangle$. Thus $|\psi_0\rangle$ will be closer to $|e\rangle$ than $|a\rangle$. Diagrammatically this can be represented as follows:

What we aim to do is take a vector $|\psi\rangle$ which will be initially equal to $|\psi_0\rangle$ and rotate it so that it ends up getting close to $|a\rangle$. For this what we would like to do is first reflect $|\psi\rangle$ about $|e\rangle$ and then about $|\psi_0\rangle$. Two reflections simultaneously correspond to a rotation. Reflecting about $|e\rangle$ can be done by considering an oracle that performs the operation

$$|x\rangle \to (-1)^{f(x)}\,|x\rangle$$



Notice that we used a similar oracle in Deustch-Josza algoritm. Let us call this operation O. The way to achieve this is to take an oracle that performs the operation

$$|x\rangle |y\rangle \to |x\rangle |y \oplus f(x)\rangle$$

The construction of this oracle was described in the section on the Deustch-Josza algoritm. Let $|y\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}}$ If $x = a$ then the oracle will perform the following operation

$$|x\rangle |y\rangle \to - |x\rangle |y\rangle$$

and for every other x as $f(x) = 0$ it will preserver the state. Thus using such an oracle and simply ignoring the $|y\rangle$ qubit will allow us to get the behaviour we desire. Now when this operation is applied on any $|\psi\rangle$ only the component along $|a\rangle$ will be flipped and the rest will remain the same. This achieves reflection about $|e\rangle$. To reflect about $|\psi_0\rangle$ we would like to use the following result that you have to prove:

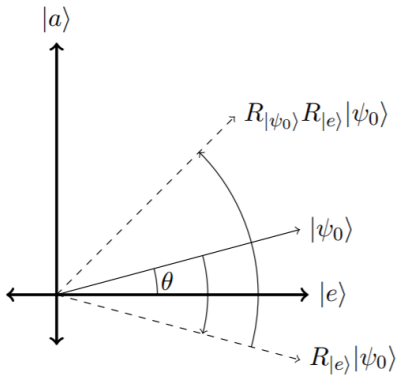
Exercise 10.2.1. Prove that the following unitary operation $U = 2 |\psi\rangle \langle\psi| - I$ achieves reflection about the state vector $|\psi\rangle$ (Hint: take any vector in the hilbert space and write it in terms of $|\psi\rangle$ and the bases of the subspace orthogonal to $|\psi\rangle$)

We define the Grover Operator G as

$$G = (2 |\psi_0\rangle \langle\psi_0| - I)O$$

This operator first reflects about $|e\rangle$ and then about $|\psi_0\rangle$ Let ψ_0 make an angle θ with $|e\rangle$ in the two

Figure 10.1: Result after applying Grover once



dimensional subspace. Then it follows that $\sin(\theta) = \frac{1}{\sqrt{N}}$. The following exercise will allow you to see how successfully applying G changes our state vector which was initially $|\psi_0\rangle$

Exercise 10.2.2. Prove by induction that

$$G^k |\psi_0\rangle = \cos((2k+1)\theta) |e\rangle + \sin((2k+1)\theta) |a\rangle$$

Thus applying G successively on $|\psi_0\rangle$ rotates the vector by an angle 2θ. We can continue applying this to $|\psi\rangle$ until the angle between $|\psi\rangle$ and $|a\rangle$ will be $\leq \theta$. Then the probability of collapsing to $|a\rangle$ will be $\geq \cos(\theta) = \frac{\sqrt{N-1}}{\sqrt{N}}$. For large N this will be very accurate. How many operations of G do we need though? Clearly we require $\left[\frac{\pi}{4\theta}\right]$ applications of G. Notice that $\theta \geq \sin(\theta) = \frac{1}{\sqrt{N}}$ and so

$$\left[\frac{\pi}{4\theta}\right] \leq \frac{\pi\sqrt{N}}{4}$$

Thus we need only $O(\sqrt{N})$ operations to achieve the task with probability $\geq \frac{\sqrt{N-1}}{\sqrt{N}}$

Thus the algorithm may be summarised as follows:

Algorithm: Grover's Search Algorithm
Input: An oracle that does the transformation

$$|x\rangle \to (-1)^{f(x)} |x\rangle$$

Output: The only a satisfying $f(a) = 1$ with probability of success $\geq \frac{\sqrt{N-1}}{\sqrt{N}}$
Procedure:
1. Create the state ψ_0 by applying $H^{\otimes n} |0\rangle$
2. Apply the Grover operation $G = (2 |\psi_0\rangle \langle\psi_0| - I)O$ $\left[\frac{\pi}{4\theta}\right]$ times
3. Measure the n qubits to get a

This case of Grover's algorithm was defined when we had to search for only one element. We can modify it when multiple x satisfied $f(x) = 1$ in which case we would have obtained different values of x as the output (only one output per iteration of the algorithm). This modification is left as an exercise. You may want to change the two vectors that we took as the basis of the two dimension subspace over which we worked. The algorithm in this case will be $O(\sqrt{N/M})$

With this we have successfully described Grover's algorithm for searching. Although the speedup is not as significant as that in fourier transform or factoring the gains are significant. Grover's algorithm can be useful in solving **NP Hard** problems by allowing us to search through the search space more quickly.

One question that may come to our minds is whether we can hope for a more significant speedup by modifying our algorithm. It has been however proven that any quantum search algorithm for the general case would still require $O(\sqrt{N})$ operations to terminate and thus Grover's algorithm is an asymptotically optimal algorithm.

Chapter 11

Physical realisation of quantum computer

11.1 DiVincenzo's criteria

According to [2], there are five (plus two) requirements for implementation of quantum compu- tation:

1. A scalable physical system with well characterized qubits
2. The ability to initialize the state of the qubits to a simple fiducial state, such as —000 . . .i
3. Long relevant decoherence times, much longer than the gate operation time
4. A "universal" set of quantum gates
5. A qubit-specific measurement capability
6. The ability to interconvert stationary and flying qubits
7. The ability faithfully to transmit flying qubits between specified locations

11.2 Harmonic oscillator quantum computer

Let us consider a very elementary system - the simple harmonic oscillator – and discuss why it does not serve as a good quantum computer. The formalism used in this example will also serve as a basis for studying other physical systems.

11.2.1 Physical Apparatus

An example of a simple harmonic oscillator is a particle in a parabolic potential well, $V(x) = \frac{m\omega^2 x^2}{2}$. In the classical world, this could be a mass on a spring, which oscillates back and forth as energy is transfered between the potential energy of the spring and the kinetic energy of the mass. It could also be a resonant electrical circuit, where the energy sloshes back and forth between the inductor and the capacitor. In these systems, the total energy of the system is a continuous parameter.

In the quantum domain, the total energy of the system can only take on a discrete set of values. An example is given by a single mode of electromagnetic radiation trapped in a high Q cavity; the total amount of energy (up to a fixed offset) can only be integer multiples of ω ,an energy scale which is determined by the fundamental constant $\hbar$ and the frequency of the trapped radiation, ω. The set of discrete energy eigenstates of a simple harmonic oscillator can be labeled as $|n\rangle$, where $n = 0, 1, 2, \ldots \infty$. The relationship to quantum computation comes by taking a finite subset of these states to represent qubits. These qubits will have

lifetimes determined by physical parameters such as the cavity quality factor Q, which can be made very large by increasing the reflectivity of the cavity walls.

11.2.2 The Hamiltonian

The Hamiltonian for a particle in a one-dimensional parabolic potential is

$$H = \frac{p^2}{2m} + \frac{m\omega^2 x^2}{2}$$

where p is the particle momentum operator, m is the mass, x is the position operator, and ω is related to the potential depth. x and p are operators in this expression, which can be rewritten as

$$H = \hbar\omega(a^\dagger a + \frac{1}{2})$$

where $a^\dagger$ and a are creation and annihilation operators, defined as

$$a = \frac{1}{\sqrt{2m\hbar\omega}}(m\omega x + ip)$$

$$a^\dagger = \frac{1}{\sqrt{2m\hbar\omega}}(m\omega x - ip)$$

The zero point energy $\hbar\frac{\omega}{2}$ contributes an unobservable overall phase factor, which can be disregarded for our present purpose. The eigenstates $|n\rangle$ of H, where $n = 0, 1, \ldots$, have the properties

$$a^\dagger a\,|n\rangle = n\,|n\rangle$$

$$a^\dagger = \sqrt{n+1}\,|n+1\rangle$$

$$a\,|n\rangle = \sqrt{n}\,|n-1\rangle$$

Later, we will find it convenient to express interactions with a simple harmonic oscillator by introducing additional terms involving a and a, and interactions between oscillators with terms such as $a_1^\dagger a_2 + a_2^\dagger a_1$. For now, however, we confine our attention to a single oscillator.

Time evolution of the eigenstates is given by solving the Schrodinger equation, from which we find that the state $|\Psi(0)\rangle = \Sigma_n c_n(0)\,|n\rangle$ evolves in time to become

$$|\Psi(t)\rangle = e^{-\frac{iHt}{\hbar}}\,|\Psi(0)\rangle = \Sigma_n c_n e^{-in\omega t}\,|n\rangle$$

11.2.3 Quantum computation

Suppose we want to perform quantum computation with the single simple harmonic oscillator described above. What can be done? The most natural choice for representation of qubits are the energy eigen states $|n\rangle$. This choice allows us to perform a controlled-NOT gate in the following way. Recall that this transformation performs the mapping on two qubit states (here, the subscript L is used to clearly distinguish 'logical' states in contrast to the harmonic oscillator basis states).

$$|00\rangle_L \rightarrow ket00_L$$

$$|01\rangle_L \rightarrow ket01_L$$

$$|10\rangle_L \rightarrow ket11_L$$

$$|11\rangle_L \rightarrow ket10_L$$

Let us encode these two qubits using the mapping

$$|00\rangle_L \rightarrow ket0$$

$$|01\rangle_L \rightarrow ket2$$
$$|10\rangle_L \rightarrow \frac{(|4\rangle + |1\rangle)}{\sqrt{2}}$$
$$|11\rangle_L \rightarrow \frac{(|4\rangle - |1\rangle)}{\sqrt{2}}$$

Now suppose that at t = 0 the system is started in a state spanned by these basis states, and we simply evolve the system forward to time $t = \frac{\pi}{\omega}$. This causes the energy eigenstates to undergo the transformation $|n\rangle \rightarrow exp(-i\pi a^\dagger a)\,|n\rangle = (-1)^n\,|n\rangle$, such that $|0\rangle, |0\rangle\, and\, |0\rangle$ stay unchanged, but $|1\rangle \rightarrow -|1\rangle$. As a result, we obtain the desired controlled-NOT gate transformation.

11.2.4 Drawbacks

There are several problems associated with the above scenario, which are :

1. Clearly, one will not always know the eigenvalue spectrum of the unitary operator for a certain quantum computation, even though one may know how to construct the operator from elementary gates. In fact, for most problems addressed by quantum algorithms, knowledge of the eigenvalue spectrum is tantamount to knowledge of the solution!
2. The technique used above does not allow one computation to be cascaded with another, because in general, cascading two unitary transforms results in a new transform with unrelated eigenvalues.
3. The idea of using a single harmonic oscillator to perform quantum computation is flawed because it neglects the principle of digital representation of information. A Hilbert space of 2^n dimensions mapped into the state space of a single harmonic oscillator would have to allow for the possibility of states with energy $2^n\hbar\omega$. In contrast, the same Hilbert space could be obtained by using n two-level quantum systems, which has an energy of at most $n\hbar\omega$.

11.3 Other implementation techniques

Physical realisation of quantum computers is a part of active research and there have been several advancements in this field. We saw that harmonic oscillator quantum computers are not a very good fit. We need more complex Hamiltonians and smarter ways to play around with the quantum mechanics. Several other examples worth noting are: Ion trap quantum computer, Opitcal photon quantum computer, Optical cavity quantum electrodynamics and also Nuclear Magnetic Resonance(NMR). You are encouraged to read more about these from the reference text.

Chapter 12

Conclusion

This marks the end of the report. I mainly described the preliminaries and basics that are needed to dive deeper into the field of quantum computing and information. **Here** is my github repository which will contain the final report with some updates and other resources including the LaTeX code that was used for making this report.

References

[1] M. A. Nielsen, Isaac L. Chuang, "Quantum Computation and Quantum Information".

[2] David P. DiVincenzo, "The Physical Implementation of Quantum Computation" (arXiv:quant-ph/0002077).

[3] https://www.qiskit.org/textbook/..