OPTIMIZATION OF QUANTUM RANDOM WALK SIMULATIONS

by

Uğur Küçük

B.S., Computer Engineering, Boğaziçi University, 2003

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Computer Engineering
Boğaziçi University

2005

ACKNOWLEDGMENTS

I would like to thank my advisor Prof. A. C. Cem Say for his invaluable guidance, support and patience during this work as well as during my undergraduate and graduate years. I am also grateful to Assist. Prof. Muhittin Mungan and Assoc. Prof. Can Özturan for their valuable criticisms and participating in my thesis jury.

I must specially thank to my family for their continuous support and trust in me. My dept to them is certainly the biggest of all.

I should not forget my workmates at Yogurt Computer Technologies whose endless cheer is one of the main contributions to my work. Thank you, comrades.

Last but not least, thank you little one, for your presence.

ABSTRACT

OPTIMIZATION OF QUANTUM RANDOM WALK SIMULATIONS

In computer science, an exponential performance gain is considered an important achievement that can extend the set of practically computable problems. Behind the interest in quantum computation, there is the fact that several quantum algorithms have been shown to provide exponential speedup against their classical counterparts. Of these, the most recent one, the one based on quantum random walks, is discussed in this work. The methods used in demonstrating the exponential algorithmic speedup by quantum random walks are analyzed in detail. This analysis comes after introductory parts where basic quantum computation concepts, quantum simulation techniques and quantum random walk ideas are discussed. A new optimization technique on the implementation of quantum random walks is also introduced. This technique is based on the idea of manipulating the order in which the constituent Hamiltonians are simulated for small durations in the iterative step of the simulation algorithm. Our approach can be generalized to optimize any quantum simulation in which the linear combination rule is used to simulate a collection of constituent Hamiltonians.

ÖZET

RASTSAL KUANTUM GEZİNTİLERİNİN BENZETİMİNİN ENİYİLENMESİ

Bilgisayar bilimleri alanında üstel ifadelerle ölçülen bir başarım artışı, pratikte çözülebilir sayılan problemler kümesinin tanımını genişletebilecek kadar önemli bir başarıdır. Kuantum bilgisayarlarına duyulan ilginin arkasında bazı kuantum algoritmalarının klasik eşdeğerleri karşısında üstel ifadelerle ölçülecek düzeyde hız kazanımları sağlamaları yatmaktadır. Bu çalışmada, bu algoritmaların en yenisi, rastsal kuantum gezintilerine dayalı olanı ele alınacak. Rastsal kuantum gezintileri yoluyla üstel algoritmik hız kazanımları elde etmenin metodları detaylı biçimde incelenecek. Bu inceleme, kuantum bilgisayarlarının temel kavramlarının, kuantum benzetim tekniklerinin ve rastsal kuantum gezintisi fikrinin tartışıldığı tanıtıcı bölümlerden sonra yer alacak. Ayrıca rastsal kuantum gezintilerinin kuramsal uygulamasına dair yeni bir eniyileme tekniği de tanıtılacak. Bu teknik benzetim algoritmasının döngüsel adımlarında küçük süreler için benzetimlenen bileşen Hamilton operatörlerinin sırası üzerinde yapılacak oynamalara dayanmaktadır. Yaklaşımımız doğrusal kombinasyonlar kuralının bir grup Hamilton operatörünün benzetiminde kullanıldığı durumlar için genelleştirilebilir.

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

Determining the problems which quantum computers can solve qualitatively faster than classical computers and constructing quantum algorithms for these problems are central issues in the field of quantum computation. There are several techniques which are known to create such performance gains, and there is a continuing research to explore new ones.

An important group of these techniques can provide sub-exponential speedups against their classical counterparts. Among these, there is Grover's famous search algorithm [1] which can provide quadratic speedup for linear search problems. A sub-exponential speedup can dramatically reduce the computational cost of some problems especially if they are of big scales. However, such techniques are of secondary importance when compared to those which can provide exponential gains.

An exponential gain is such an important achievement in computer science that it can change the known contents of the set of practically computable problems. The existence of such quantum algorithms which can provide exponential speedup over their classical counterparts is known since the early 1990's. In particular, Shor's algorithm for factorization [2] is one of the biggest achievements in the field of quantum computation. There are several other problems for which quantum algorithms have been demonstrated to be exponentially faster than the classical ones. However, it turns out that these problems are strongly related and the quantum algorithms devised for them depend on similar principles. To be more accurate, until very recently, exponential speedup had been demonstrated only for those problems which can be formulated as a hidden subgroup problem and only by those algorithms, which depend somehow on the quantum Fourier transformation method.

A recent result challenges this picture. A quantum algorithm, which does not involve the quantum Fourier transformation and yet is able to provide exponential speedup over its classical counterparts was demonstrated in [3]. The problem targeted with this algorithm is a

black box problem defined on a graph theoretical framework. The methods used to solve this problem mainly depend on a quantum version of the random walk techniques and the concept of quantum simulation is used to implement this idea.

The main objective of the current work is to develop an understanding of the exponential speedup provided by quantum random walks, which is one of the most important and promising results recently demonstrated in the field of quantum algorithms. The techniques used in [3] will be examined in detail and an improvement in the quantum walk implementation will be proposed. The main constraint in doing these is being accessible by an average computer science person whose degree of familiarity with the topic is assumed to be low.

Here is how the rest of this thesis is structured. Chapter 2 serves as a brief introduction to quantum computation for those who have little (if any) familiarity with the field. In Chapter 3, the concept of quantum simulation will be analyzed and some of the simulation techniques to be used in later chapters will be introduced. Chapter 4 is a survey of quantum walks. It aims to provide an understanding of how quantum versions of random walks are implemented and lists some of the important results related to quantum walks. It is Chapter 5, where a discussion of the methods used to demonstrate the exponential speedup by quantum walks can be found. In Chapter 6, we will propose an improvement on the implementation of quantum walks used in [3], and Chapter 7 will be a conclusion.

2. BASICS OF QUANTUM COMPUTATION

Two-valued classical logic can be viewed as the model for the classical theory of computation in the sense that in both frameworks, complex systems are built upon two-valued atomic units and various relations among them. A reason for this choice is that people are used to think in the same manner and another is the ease of implementation of such two valued models in practical computational devices. Various technologies from vacuum tubes to microchips have been used for this purpose up to now, and most probably new technologies are to be added to the list in the future. What makes quantum computation a challenge for us is the projection that in the not so far future, it will be easy to build computational devices which enjoy the advantages presented by the quantum mechanical laws. We will be calling such devices quantum computers not because they are (or will be) the only ones which obey the laws of quantum mechanics, but to indicate that they are the ones which find their computational model in quantum mechanics. Once quantum computers come to be a part of reality, there will be the need for an understanding of that quantum mechanical model of computation. This is what the field of quantum computation tries to build and here, the basics of this field are presented.

2.1. Hilbert Space and Quantum Bits

The central position that a "bit" (Shannon bit) occupies in the classical theory of computation is reserved for what is called a qubit (quantum bit) in quantum computation. Hence, a qubit can be said to be the basic unit of information for quantum computers. In practice, a qubit can be any two state quantum system such as an electron with two spin states or a hydrogen atom with two energy levels. (For a more exhaustive list, see [4].) Most of the literature in the field of quantum computation refers to an abstract mathematical model of a qubit rather than specific physical implementations of it. This will be the way to be followed here. Building a formal mathematical definition for a qubit requires the introduction of the concept of a Hilbert space which is often found useful for describing the states of quantum

systems. (For more detailed but still brief introductions to the concept of a Hilbert space and its use in quantum mechanics, see [5, 6].)

Formally, a Hilbert Space H is an inner product space that is complete with respect to the norm defined by the inner product. By an inner product space, we mean a vector space over the complex numbers with an inner product $(\cdot,\cdot): H \times H \to C$ satisfying

i) $(u,u) \geq 0$ with equality if and only if $u = 0$

ii) $(u, v+w) = (u,v) + (u,w)$ and $(u, \lambda v) = \lambda (u,v)$

iii) $(u,v) = (v,u)^*$, where * denotes the complex conjugate.

H is also required to be complete with respect to the norm $\|\cdot\|$ over this space, which is defined as $\|u\| = \sqrt{(u,u)}$.

Although Hilbert space is also useful when handling infinite dimensional vector spaces, quantum computation mostly deals with finite dimensional ones. (All *n*-dimensional Hilbert spaces are isomorphic, hence they can be denoted by H_n.) In quantum mechanics, the dimension of a Hilbert space is a function of the number of observable states of the quantum system which that Hilbert space is used to represent. In general, an *n*-dimensional Hilbert space H_n is a space of *n*-tuples of complex numbers. These n-tuples are called state vectors. The state of a quantum system at a time is represented by one such member of the Hilbert space of the appropriate size. State vectors are usually denoted by column vectors for which, following the Dirac notation of vectors, we use the ket ($|label\rangle$) formalism as in (2.1). Note that *u* is just some label and u_1, u_2,.., u_n are complex numbers.

$$u = |u\rangle \in H_n \text{ and } |u\rangle = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix} \tag{2.1}$$

Two kets $|u\rangle$ and $|w\rangle$ represent the same state if they differ by a non-zero multiplicative constant: $|u\rangle = \lambda|w\rangle$. ($|u\rangle$ and $|w\rangle$ are said to be the same up to a global phase factor λ.) In general, it is a convention to use a ket vector of unit length to represent the state of a quantum system and this convention is followed in the present context.

What should immediately follow the ket vectors is the definition of the bra ($\langle label|$) vectors. It is convenient in this domain to define bra vectors as the Hermitian conjugates (complex conjugate transpose will be denoted by superscript dagger †.) or adjoint vectors of the members of a Hilbert space. Therefore, for $u \in H_n$, the adjoint $v = u^\dagger$ is a row vector as shown in (2.2). Again note that *v* is just some label and $v_1, v_2, .. v_n$ are complex numbers.

$$v = \langle v| = \begin{bmatrix} v_1 & v_2 & \cdots & v_n \end{bmatrix} = u^\dagger = \left(|u\rangle^*\right)^T = \begin{bmatrix} u_1^* & u_2^* & \cdots & u_n^* \end{bmatrix} \tag{2.2}$$

Now we have bras and kets so that it is time to form the bra(c)ket ($\langle label1|label2\rangle$) which is the origin of the naming of vectors in Dirac notation. A bracket is a natural way to denote the inner product and hence it stands for a complex number. Therefore, for $w, z \in H_n$, the inner product of *w* and *z* is a complex number as shown in (2.3).

$$(w, z) = w^\dagger z = \langle w \| z \rangle = \langle w | z \rangle = \sum_{i=1}^{n} w_i^* z_i \in C \tag{2.3}$$

So much mathematical formalism is enough that we can start to build our definition of a qubit. The qubit is an inheritor of the classical bit in many aspects. Just like a classical bit, a

qubit is a two state system in principle. These states, following the tradition, are labeled by 0 and 1, and they correspond to two different (observable) physical states of the system, no matter which physical implementation is used. The essential difference is that a qubit is a quantum system. So unlike a classical bit, a qubit is not restricted to be in one of these two principal states. Instead it is generally in a linear combination (or superposition) of them as quantum mechanics suggests. Therefore the state of a qubit, $|\psi\rangle$, is a vector in a two dimensional complex vector space where the principal states are natural candidates to serve as a basis. We tend to call this basis the computational basis of a qubit and the complex vector space this basis spans is in fact a two dimensional Hilbert space. So a qubit, in formal mathematical terms, can be defined to be a quantum system whose state lies in a two dimensional Hilbert space [7].

The computational basis that spans the state space of a qubit is shown in (2.4).

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \text{ and } |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{2.4}$$

The general state of a qubit, $|\psi\rangle$, is a linear combination or superposition of the base states. Note that α and β are complex numbers so the overall state is a member of the two dimensional complex vector space or H_2.

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle = \alpha\begin{bmatrix} 1 \\ 0 \end{bmatrix} + \beta\begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha \\ \beta \end{bmatrix} \in H_2 \tag{2.5}$$

Since the ket vectors we use are of unit length, we can assert the following.

$$|\alpha|^2 + |\beta|^2 = 1 \tag{2.6}$$

In the case of a classical bit, the state is identical to the outcome of a measurement that is performed to retrieve the content. On the other hand, what one gets out of measuring a qubit is not generally equivalent to the state of it. The quantum mechanical explanation of the action of measurement is not a simple one. (A separate section will be devoted to this topic.) In the simplest terms, one can assert that measuring a qubit which is prepared to be in the state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ in the computational basis, has two possible outcomes. These are what we have been calling the principal states. The outcome of a measurement is $|0\rangle$ with probability $|\alpha|^2$ and it is $|1\rangle$ with probability $|\beta|^2$. These probabilities sum up to one by (2.6).

$$\text{probability of observing 0: } \mathrm{P}(|0\rangle) = |\alpha|^2 \tag{2.7}$$

$$\text{probability of observing 1: } \mathrm{P}(|1\rangle) = |\beta|^2 \tag{2.8}$$

Note that, in case of a ket vector which was not of unit length, the above equalities would not hold. Instead one could say the probability of observing $|0\rangle$ and $|1\rangle$ are proportional to $|\alpha|^2$ and $|\beta|^2$ respectively. This should be why ket vectors of unit length are more favorable than the others.

Another important point is that after such a measurement, the state of the qubit settles on the observed value so that if it is measured again, it is guaranteed that the result remains the same. Therefore, for practical purposes it can be stated that measurements cause changes on the state of the qubits provided that they are not in one of their base states.

It is straightforward to state that a qubit can be used to encode a much larger amount of information than that a classical bit can. Although the space of two dimensional unit ket vectors seems to be restricted, still (uncountably) infinitely many of these vectors are available. So in principle a qubit can store infinite amount of information but in practice, the important thing is how much one can retrieve from it. The task of preparing a qubit (or rather a

set of qubits) in a state that makes it possible to gather the information one seeks, belongs to the quantum algorithms which will be examined soon.

As a last issue, let us consider the so-called Bloch sphere representation of the state of a qubit, which is of use when to visualise the state of a qubit and the effect of the operators on it. This representation maps the state of a qubit to a point on the surface of a unit sphere. Here a derivation of the model is to be given. To begin with, we have the general state vector of a qubit $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ and the normalisation constraint $|\alpha|^2 + |\beta|^2 = 1$ where α and β are complex numbers. If α and β are expressed in polar coordinates, we then have the following representation of the general state of a qubit.

$$|\psi\rangle = r_\alpha(\cos\theta_\alpha + i\sin\theta_\alpha)|0\rangle + r_\beta(\cos\theta_\beta + i\sin\theta_\beta)|1\rangle = r_\alpha e^{i\theta_\alpha}|0\rangle + r_\beta e^{i\theta_\beta}|1\rangle \tag{2.9}$$

We had seen that two kets represent the same state if they differ by a global phase factor. So we are free to multiply $|\psi\rangle$ with $e^{-i\theta_\alpha}$ to get $|\psi'\rangle$ which is equal to $|\psi\rangle$ up to the global phase factor, $e^{-i\theta_\alpha}$, that has no observable effects.

$$|\psi'\rangle = r_\alpha|0\rangle + r_\beta e^{i(\theta_\beta - \theta_\alpha)}|1\rangle = r_\alpha|0\rangle + r_\beta e^{i\theta}|1\rangle \tag{2.10}$$

Now, writing the coefficient of $|1\rangle$ again in Cartesian coordinates, we get,

$$|\psi'\rangle = r_\alpha|0\rangle + (x + iy)|1\rangle. \tag{2.11}$$

Then by the normalisation constraint, we have the following equation.

$$\langle\psi'|\psi'\rangle = (r_\alpha)^2 + (x + iy)^*(x + iy) = {r_\alpha}^2 + x^2 + y^2 = 1 \tag{2.12}$$

The equation in (2.12) is the equation for a unit sphere in the real 3-dimensional space with the Cartesian coordinates x, y, r_α. Renaming the third coordinate as z gives the coordinate system x, y, z. The same space can also be defined by the polar coordinates r, θ, ϕ. Figure 2.1 demonstrates the relation between the Cartesian coordinates x, y, z and the polar coordinates r, θ, ϕ which define the same point on the 3-dimensional space.

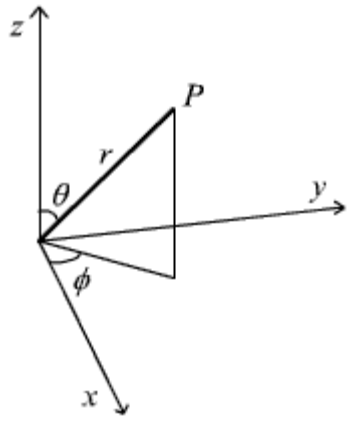

Figure 2.1. Cartesian and polar coordinates for the 3-dimensional space

As Figure 2.1 suggests we have the following transformations

$$x = r \sin\theta \cos\phi \tag{2.13}$$

$$y = r \sin\theta \sin\phi \tag{2.14}$$

$$z = r \cos\theta \tag{2.15}$$

Using the above rules with $r = \sqrt{x^2 + y^2 + z^2} = 1$ and using z for r_α, (2.11) can be transformed to,

$$\begin{aligned} |\psi'\rangle &= \cos\theta |0\rangle + (\sin\theta \cos\phi + i \sin\theta \sin\phi) |1\rangle \\ &= \cos\theta |0\rangle + e^{i\phi} \sin\theta |1\rangle. \end{aligned} \tag{2.16}$$

Now, we have only two real parameters θ and ϕ defining the state $|\psi'\rangle$. The term $e^{i\phi}$ is called a relative phase factor and unlike the global phase, it has important effects. Note that, for $\theta = 0$, $|\psi'\rangle$ turns out to be $|0\rangle$ and for $\theta = \pi/2$, $|\psi'\rangle$ turns out to be $e^{i\phi}|1\rangle$. So our mapping of the state of a qubit is not yet covering a full sphere. Instead, it covers the upper half of it. We get over this problem by a small adjustment: We let the point $(1, \theta, \phi)$ on the sphere to denote the state $|\psi\rangle = \cos\frac{\theta}{2}|0\rangle + e^{i\phi}\sin\frac{\theta}{2}|1\rangle$ (instead of $|\psi\rangle = \cos\theta|0\rangle + e^{i\phi}\sin\theta|1\rangle$) hence, the state space of a qubit covers the full surface of the unit sphere. The Cartesian coordinates $(\sin\theta\cos\phi, \sin\theta\sin\phi, \cos\theta)$ of the point is called the Bloch vector. This completes our definition of the Bloch sphere.

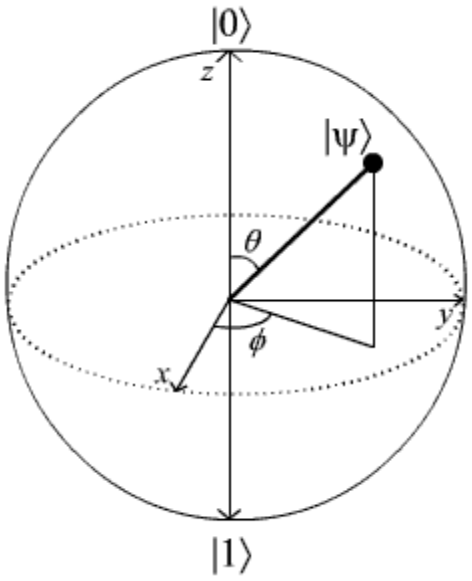

Figure 2.2. The Bloch sphere representation of the state of a qubit

Note that $0 \le \theta \le \pi$ and $0 \le \phi \le 2\pi$ are the valid coordinates on the Bloch sphere. The θ coordinate determines the probabilities for observing $|0\rangle$ and $|1\rangle$. There are infinitely many points on the surface of the sphere for each value of $0 \le \theta \le \pi$ with the exception of the endpoints where the second coordinate ϕ is practically meaningless. The ϕ coordinate is

called the phase angle and it determines the relative phase factor and hence the ratio of the magnitudes of real and imaginary parts in the amplitude of $|1\rangle$.

The Bloch sphere representation is useful when analyzing the state of a qubit and the effects of the operators on it. However there is no simple generalization of this model to systems of more than one qubits, and hence its uses are limited.

2.2. Tensor Product and Quantum Registers

A quantum register is simply a collection of qubits and hence it is a multiparticle quantum system. To understand the state and behavior of such a system, there is the need for a mathematical tool for putting vector spaces together to form larger vector spaces. This task is assigned to the tensor product operator ($\otimes$).

The tensor (or Kronecker) product of two matrices A and B, where *A* is *m* by *n* and *B* is *r* by *s* is a matrix *mr* by *ns*. The product is calculated as in (2.17).

$$A \otimes B = \begin{pmatrix} a_{11} & \cdots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nm} \end{pmatrix} \otimes B = \begin{pmatrix} a_{11}B & \cdots & a_{1m}B \\ \vdots & \ddots & \vdots \\ a_{n1}B & \cdots & a_{nm}B \end{pmatrix} \tag{2.17}$$

The tensor product $H \otimes K$ of two Hilbert spaces *H* of dimension *m* and *K* of dimension *n* is a Hilbert space of dimension *m* times *n*. The members of $H \otimes K$ are linear combinations of the tensor products of the members of *H* and the members of *K*. In other words if $h_1, h_2, .., h_m$ is a base for *H* and $k_1, k_2, .., k_n$ is a base for *K* then $\{h_i \otimes k_j \mid 1 \le i \le m, 1 \le j \le n\}$ is a base for $H \otimes K$.

An informal abstraction for $H \otimes K$ can be constructed by the following properties it has:

1. $\lambda(h_1 \otimes k_1) = (\lambda h_1) \otimes k_1 = h_1 \otimes (\lambda k_1)$, for all $\lambda \in C$, $h_1 \in H$, $k_1 \in K$,
2. $(h_1 + h_2) \otimes k_1 = (h_1 \otimes k_1) + (h_2 \otimes k_1)$, for all $h_1 \in H$, $h_2 \in H$, $k_1 \in K$,
3. $h_1 \otimes (k_1 + k_2) = (h_1 \otimes k_1) + (h_1 \otimes k_2)$, for all $h_1 \in H$, $k_1 \in K$, $k_2 \in K$,

For state vectors *u* and *w* there are several ways to denote the tensor product $u \otimes w$:

$$u \otimes w = |u\rangle \otimes |w\rangle = |u\rangle|w\rangle = |uw\rangle. \tag{2.18}$$

Most generally, a quantum register is a collection of *n* qubits, each with their individual computational basis of $B_i = \{|0\rangle, |1\rangle\}$ for $1 \le i \le n$, which spans their individual state spaces. Then the state of the overall system lies in a 2^n dimensional Hilbert space *H*, which is the tensor product of these two dimensional Hilbert spaces $H_1, H_2, .., H_n$ for each individual qubit. The computational basis of this system is $B = \{b_1 \otimes b_2 \otimes .. \otimes b_n \mid b_i \in B_i, 1 \le i \le n\}$, hence it contains 2^n states of the form $|\{0,1\}^n\rangle$. The general state of the system is described by a unit ket vector which is a linear combination of the members of this computational basis. The probability of observing the system in a base state is again related to the square of the amplitude for that base state in the general state vector.

In particular, a quantum register of two qubits is a system whose state lies in a $2^2 = 4$ dimensional Hilbert space. The computational basis for this 4-dimensional Hilbert space is $\{0,1\}^n = \{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$, where the base states are given by

$$|00\rangle = |0\rangle \otimes |0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \tag{2.19}$$

$$|01\rangle = |0\rangle \otimes |1\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \tag{2.20}$$

$$|10\rangle = |1\rangle \otimes |0\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \tag{2.21}$$

$$|11\rangle = |1\rangle \otimes |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}. \tag{2.22}$$

The general state vector ψ of a quantum register of two qubits is therefore a linear combination of these four base states. Once again $\alpha_0, \alpha_1, \alpha_2, \alpha_3$ are complex numbers and since kets of unit length are preferred, the sum of their squares $|\alpha_0|^2 + |\alpha_1|^2 + |\alpha_2|^2 + |\alpha_3|^2$ is equal to one.

$$|\psi\rangle = \alpha_0|00\rangle + \alpha_1|01\rangle + \alpha_2|10\rangle + \alpha_3|11\rangle = \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix}, \tag{2.23}$$

$$|\alpha_0|^2 + |\alpha_1|^2 + |\alpha_2|^2 + |\alpha_3|^2 = 1 \tag{2.24}$$

The state of the two-qubit register can be observed to be in one of the four base states with the following probabilities.

$$\text{probability of observing 00: } \mathrm{P}(|00\rangle) = |\alpha_0|^2, \tag{2.25}$$

$$\text{probability of observing 01: } \mathrm{P}(|01\rangle) = |\alpha_1|^2, \tag{2.26}$$

$$\text{probability of observing 10: } \mathrm{P}\left(|10\rangle\right) = |\alpha_2|^2 , \tag{2.27}$$

$$\text{probability of observing 11: } \mathrm{P}\left(|11\rangle\right) = |\alpha_3|^2 . \tag{2.28}$$

Let us now consider a register of two qubits, whose individual states are known to be $|\psi\rangle = \alpha_0|0\rangle + \alpha_1|1\rangle$ and $|\theta\rangle = \beta_0|0\rangle + \beta_1|1\rangle$. Then the state of the overall register $|\psi\theta\rangle$ is given by the tensor product of the two individual states as in (2.29).

$$\begin{aligned} |\psi\rangle \otimes |\theta\rangle &= \left(\alpha_0|0\rangle + \alpha_1|1\rangle\right) \otimes \left(\beta_0|0\rangle + \beta_1|1\rangle\right) \\ &= \alpha_0\beta_0|00\rangle + \alpha_0\beta_1|01\rangle + \alpha_1\beta_0|10\rangle + \alpha_1\beta_1|11\rangle = \begin{bmatrix} \alpha_0\beta_0 \\ \alpha_0\beta_1 \\ \alpha_1\beta_0 \\ \alpha_1\beta_1 \end{bmatrix} \end{aligned} \tag{2.29}$$

The reader should be able to generalize the constructions above to quantum registers of larger size by the help of a straight analogy. As a last remark, we should note that it might be impossible to represent the state of a composite system as the product of the states of the individual components of it. Such states are called entangled. This issue will be examined in a separate section. For now just consider the following state which is a well known example of entangled states.

$$|\psi\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{2.30}$$

2.3. The Time Evolution and Unitary Transformations

In order to understand how to store information into qubits, manipulate their content and retrieve the useful data within them, we need to know in which ways the state of a quantum

system changes. In quantum mechanics the changes in the state of an isolated quantum system which is not subjected to observation are closely related to the total energy of this system.

A Hermitian (self-adjoint) operator called the Hamiltonian (H) can be said to govern the evolution of the state of a quantum system. The eigenvalues and the eigenvectors of the Hamiltonian characterize the stationary energy states (or basis states) of the system, hence the Hamiltonian reflects the dynamical properties of the system it belongs to. (This means the Hamiltonian itself may be changing with time as the system changes.) The evolution is described by the Schrödinger equation,

$$i\hbar \frac{d}{dt}|\psi\rangle = H|\psi\rangle. \tag{2.31}$$

where $i = \sqrt{-1}$ and $\hbar$ is Planck's constant divided by 2π . It is common practice to write the equation as in (2.32), where $\hbar$ is absorbed into the Hamiltonian H .

$$i \frac{d}{dt}|\psi\rangle = H|\psi\rangle. \tag{2.32}$$

This continuous picture of the time evolution can be transformed to a discrete picture of the same phenomenon which is of more use for computational needs. Consider the general solution to Schrödinger's equation which can be verified to be

$$|\psi(t)\rangle = e^{-i(Ht+c)}|\psi(0)\rangle, \tag{2.33}$$

where *c* is an arbitrary constant. Then the state of the system at discrete time points t_1 and t_2 can be related as follows.

$$|\psi_2\rangle = e^{-iH(t_2 - t_1)}|\psi_1\rangle \tag{2.34}$$

Before proceeding we should refresh some basic linear algebraic facts. Recall that, those values $\lambda_1, \lambda_2, ..$ which satisfy the equation $A|a\rangle = \lambda|a\rangle$ for some vector $|a\rangle$ are called the eigenvalues of the operator A and those vectors $|a_1\rangle, |a_2\rangle, ...$ which satisfy $A|a\rangle = \lambda_i|a\rangle$ are called the eigenvectors of the operator A corresponding to the eigenvalue λ_i. An important representation theorem, known as spectral decomposition, makes use of these definitions to represent a normal operator A, in the form $\sum_i \lambda_i |X_i\rangle\langle X_i|$, where a normal operator is one that satisfies $AA^\dagger = A^\dagger A$. Then any function of $f(A)$ of the operator A, can be rewritten as $f(A) = \sum_i f(\lambda_i)|X_i\rangle\langle X_i|$. Those matrices which satisfy $A^\dagger = A$ are called Hermitian, and they are trivially normal. The decomposition $A = \sum_i \lambda_i |X_i\rangle\langle X_i|$ leads to a form $A = U\Lambda U^\dagger$ where Λ is a diagonal matrix with eigenvalues λ_i of A on its diagonal entries and U is a matrix whose columns are the normalized eigenvectors of A in the corresponding order. Since a Hermitian matrix is known to have a complete set of orthonormal eigenvectors, it turns out that the columns of U are orthonormal. Hence, it exhibits the interesting property $UU^\dagger = U^\dagger U = I$, which means U is a unitary matrix. We will soon be listing some properties of unitary matrices, for now let us return to the original discussion. (For more linear algebraic topics, see [8].)

By the spectral decomposition theorem, we can write $e^{iA} = e^{i\lambda_1}|x_1\rangle\langle x_1| + .. + e^{i\lambda_n}|x_n\rangle\langle x_n|$ and $\left(e^{iA}\right)^\dagger = e^{-i\lambda_1}|x_1\rangle\langle x_1| + .. + e^{-i\lambda_n}|x_n\rangle\langle x_n|$ for an ordinary operator A. If A is a Hermitian operator then its eigenvecors are orthonormal (i.e. $|x_i\rangle\langle x_j| = 0$, for $i \neq j$) and hence it turns out that $\left(e^{iA}\right)\left(e^{iA}\right)^\dagger = \left(e^{iA}\right)^\dagger\left(e^{iA}\right) = I$. Therefore the term e^{iA} is equivalent to a unitary operator if A is Hermitian. Then the term $e^{-iH(t_2 - t_1)}$ in (2.34) is indeed equivalent to a unitary operator which is a function of t_1 and t_2:

$$U(t_1, t_2) = e^{-iH(t_2 - t_1)} \tag{2.35}$$

Then the equation in (2.34) becomes,

$$|\psi_2\rangle = U(t_1, t_2)|\psi_1\rangle \tag{2.36}$$

This result is important in several ways. First of all, it establishes a one to one correspondence between the continuous (which employs the Schrödinger equation and Hamiltonians) and discrete (which employs unitary operators) interpretations of quantum dynamics. So it can safely be stated that closed quantum systems can be transformed only via unitary transformations. However, quantum computation requires opening up the closed systems to retrieve the information within them. Hence, the continuous evolution governed by the Hamiltonian operators and the Schrödinger equation is not the only way the state of a quantum system changes in time. In cases where a quantum system is measured, the state of the system is said to perform a discontinuous and sudden jump into a basis state depending on the result of the measurement. The nature of quantum measurements and unitary transformations will be examined in the following sections.

2.4. The Unitary Transformations and Quantum Gates

A classical computational device can be said to be made up of logic gates and the links between them. Each logic gate is assigned a functionality according to which it transforms its inputs to produce the desired outputs. Every classical computational process should be expressible in terms of these unit functionalities. It is possible to develop an analogous formalism for quantum computation which employs the concept of quantum gates.

A quantum gate U is simply a linear operator whose input $|\psi_1\rangle$ and output $|\psi_2\rangle$ are members of the same Hilbert space and when U is applied over $|\psi_1\rangle$ the result is $|\psi_2\rangle$. As it was shown in the previous section the transformation, applied to the input ket vector to

produce the output ket vector, has to be a unitary transformation. This is the only constraint on quantum gates. Therefore, any *m* by *m* unitary matrix defines a valid quantum gate on an *m*-dimensional Hilbert space. The action of the gate U over the state vector $|\psi_1\rangle$ is simply denoted by the product of the matrix and the state vector.

$$U|\psi_1\rangle = |\psi_2\rangle \tag{2.37}$$

A unitary operator is also linear on the bra vectors, although this is sometimes hard to visualize. The effect of a general linear operator *U,* on a bra vector can be defined as in (2.38).

$$\langle\psi|U = \left(U^\dagger|\psi\rangle\right)^\dagger \tag{2.38}$$

Then, the following equalities become straightforward.

$$\langle\psi|U|\lambda\rangle = (\psi, U\lambda) = \left(U^\dagger\psi, \lambda\right) \tag{2.39}$$

It is useful to state that every unitary matrix U has an inverse, which is in fact equal to the adjoint of it, $UU^\dagger = U^\dagger U = I$. The adjoint operator $U^\dagger$ is also unitary, therefore it defines another valid quantum gate.

$$U^{-1} = U^\dagger \tag{2.40}$$

Then, every quantum computational process can be inverted. The issue of the reversibility of quantum computation will be handled soon. For now, let us examine some further features of unitary operators. It can be derived from (2.40) that, a unitary operator preserves the inner products and hence the lengths and of the vectors on which it is applied. For unitary operator *U* and the vectors $|v\rangle$ and $|w\rangle$ let $U|v\rangle = |v'\rangle$ and $U|w\rangle = |w'\rangle$. Then the inner products $\langle v'|w'\rangle$ and $\langle v|w\rangle$ are equivalent by

$$\langle v'|w'\rangle = \big(U|v\rangle\big)^{\dagger}\big(U|w\rangle\big) = \langle v|U^{\dagger}U|w\rangle = \langle v|w\rangle \tag{2.41}$$

In particular, if $|v\rangle = |w\rangle$ we see that the length of a vector $|v\rangle$ is also preserved under *U*.

$$\big\||v'\rangle\big\| = \sqrt{\langle v'|v'\rangle} = \sqrt{\langle v|v\rangle} = \big\||v\rangle\big\| \tag{2.42}$$

The reader should also note that two unitary operators *U* and *V* can be taken to be equivalent if they differ only by a multiplicative constant $\lambda = e^{i\theta}$. Let $V = \lambda U$. *U* maps the state $|w\rangle$ to $U|w\rangle$ while *V* maps the same state to $V|w\rangle = \lambda U|w\rangle$. Since $U|w\rangle$ and $\lambda U|w\rangle$ represent the same state *U* and *V* are said to be equivalent.

In general, a quantum gate with *n*-bit input can be modeled by a 2^n by 2^n unitary matrix. This matrix acts linearly on each basis vector of the 2^n dimensional Hilbert space. Hence describing how a quantum gate effects the 2^n members of the computational basis is sufficient to describe its effect on the overall space. If the effect of an *n*-bit quantum gate *U* on the 2^n members of the computational basis is given as $|k'\rangle = U|k\rangle$ for , $0 \le k \le 2^n - 1$ then the unitary matrix *U* describing the gate is given by the following equation.

$$U = \sum_{k=0}^{2^n-1} |k'\rangle\langle k| \tag{2.43}$$

In circuit notation, a quantum gate is symbolized by a box with the name of the gate as its label. The straight lines from left to right should be taken to symbolize the passage of time rather than physical wires. (Other elements of the circuit notation will be introduced as the need arises in the context.) The computation $U|\psi_1\rangle = |\psi_2\rangle$ can be represented as in Figure 2.3.

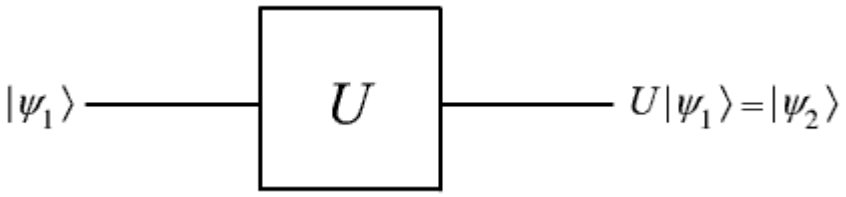

Figure 2.3. The circuit representation for the computation $U|\psi_1\rangle = |\psi_2\rangle$

The expression $UV|\theta_1\rangle$ defines the state of a register which was initially at state $|\theta_1\rangle$ and to which the gates V and U are succesively applied. The circuit representation for this computation is given in Figure 2.4. Care should be taken about the order of the operators being applied.

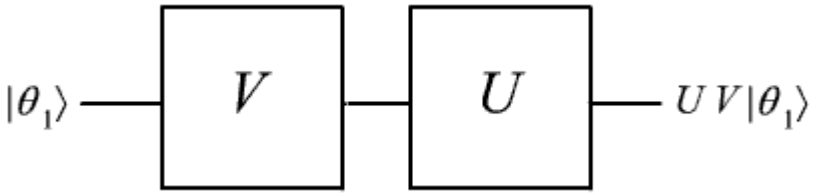

Figure 2.4. The circuit representation for the computation $UV|\theta_1\rangle$

Sections 2.4.1 and 2.4.2 will introduce the basics of single and multi-bit quantum gates. The issue of reversibility is discussed in Section 2.4.3 and the issue of universality for quantum gates is discussed in Section 2.4.4.

2.4.1. Single Qubit Quantum Gates

A single qubit quantum gate U is simply a unitary mapping defined on the two dimensional Hilbert Space, H_2. Its linear effect on the computational basis has the following form,

$$U|0\rangle = a|0\rangle + b|1\rangle \tag{2.44}$$

$$U|1\rangle = c|0\rangle + d|1\rangle \tag{2.45}$$

Then its effect on a general state vector, $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, can be shown as,

$$\begin{aligned} U|\psi\rangle &= \alpha U|0\rangle + \beta U|1\rangle \\ &= \alpha\left(a|0\rangle + b|1\rangle\right) + \beta\left(c|0\rangle + d|1\rangle\right) \\ &= \left(\alpha a + \beta c\right)|0\rangle + \left(\alpha b + \beta d\right)|1\rangle \end{aligned} \tag{2.46}$$

This operator can be represented by a 2 by 2 unitary matrix as in (2.47). Note the relation between the columns of the matrix and the equations in (2.46).

$$U = \begin{bmatrix} a & c \\ b & d \end{bmatrix} \tag{2.47}$$

Hence, the product of the matrix and the state vector is another way to formulate the effect of the operator on the state of a qubit.

$$U|\psi\rangle = \begin{bmatrix} a & c \\ b & d \end{bmatrix}\begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \alpha a + \beta c \\ \alpha b + \beta d \end{bmatrix} = \left(\alpha a + \beta c\right)|0\rangle + \left(\alpha b + \beta d\right)|1\rangle \tag{2.48}$$

The Bloch sphere formalism can be useful to visualize the effect of a single qubit operator. We had seen that the state of a qubit could be represented as a point on the surface of the Bloch sphere. The application of a unitary operator moves this state to another point on the surface. Then the Bloch vector can be said to be rotated about an axis n and through an angle θ by the unitary operator. This picture will soon be useful to define how to implement an arbitrary unitary operator.

Let us now consider some of the most fundamental single-bit quantum gates in more detail. The Hadamard gate must be the most famous quantum gate and the Pauli operators should be the most useful ones to help our intuition. Let us begin with the Hadamard gate or the Walsh-Hadamard gate as it is sometimes referred. (The Hadamard gate is often symbolized by the capital letter *H*, which had been previously used to denote the Hilbert spaces and the

Hamiltonian operators. The reader should be able to distinguish between these, with the help of the context.)

$$H = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{2.49}$$

When applied to a base state, the Hadamard gate generates an equal superposition of the members of the computational basis. Measuring these states returns $|0\rangle$ with probability 0.5 and returns $|1\rangle$ with probability 0.5.

$$H|0\rangle = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle \tag{2.50}$$

$$H|1\rangle = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}\begin{bmatrix} 0 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ -1 \end{bmatrix} = \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle \tag{2.51}$$

Many quantum algorithms start with the preparation of the equal superposition with help of the Hadamard gates. After this step, a so-called parallel computation can be carried over each base state. The effect of the Hadamard gate can be visualized as a shift in the computational basis. Or in other words, the Hadamard gate defines an alternative basis for computation (the Hadamard basis): $\{|+\rangle,|-\rangle\}$ where $|+\rangle = H|0\rangle$ and $|-\rangle = H|1\rangle$. (Note that the states $|+\rangle$ and $|-\rangle$ are the poles on the x-axis of the Bloch sphere.) In order to return to the original basis, again Hadamard gates should be used. Note that H is both Hermitian and unitary so $H^2 = I$.

An interesting group of one-bit quantum gates is called the Pauli matrices or Pauli spin matrices. These gates are the Pauli X (σ_x), Pauli Y (σ_y) and Pauli Z (σ_z).

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, \quad Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \tag{2.52}$$

The Pauli X operator, in particular, can be seen as an analogue of the classical NOT gate when it is applied to a base state. When it is applied to a superposition state it simply swaps the amplitudes for the base states as a result of linearity.

$$X|0\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = |1\rangle \tag{2.53}$$

$$X|1\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} = |0\rangle \tag{2.54}$$

The Pauli Z operator is also known as the phase-flip gate. It applies a phase factor of -1 to the state $|1\rangle$ and leaves $|0\rangle$ unchanged. Its effect on a general state can again be deduced from linearity.

$$Z|0\rangle = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} = |0\rangle \tag{2.55}$$

$$Z|1\rangle = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}\begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ -1 \end{bmatrix} = -|1\rangle \tag{2.56}$$

In the Hadamard basis, the Pauli Z operator acts as a not gate.

$$Z|+\rangle = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}\frac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ -1 \end{bmatrix} = |-\rangle \tag{2.57}$$

$$Z|-\rangle = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}\frac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ -1 \end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ 1 \end{bmatrix} = |+\rangle \tag{2.58}$$

Together with the 2 by 2 identity matrix, the Pauli matrices form an orthogonal basis for the space of 2 by 2 matrices. So, any two-dimensional square matrix U can be uniquely

defined in the following form where u_t, u_x, u_y, u_z are complex numbers and I is the two dimensional identity operator.

$$U = u_t I + u_x X + u_y Y + u_z Z \tag{2.59}$$

The Pauli X, Y, Z operators are so called because when they are exponentiated they define the rotation operators R_x, R_y, R_z about the x-, y-, z-axes of the Bloch sphere. Let us consider the power series expansion of $e^{i\theta A}$ for an arbitrary operator A and the real angle θ.

$$\begin{aligned} e^{i\theta A} &= I + i\theta A + \frac{(i\theta A)^2}{2!} + \frac{(i\theta A)^3}{3!} + \frac{(i\theta A)^4}{4!} + \frac{(i\theta A)^5}{5!} + \ldots \\ &= I + i\theta A - \frac{(\theta A)^2}{2!} - i\frac{(\theta A)^3}{3!} + \frac{(\theta A)^4}{4!} + i\frac{(\theta A)^5}{5!} + \ldots \end{aligned} \tag{2.60}$$

Now consider the case $A^2 = I$.

$$\begin{aligned} e^{i\theta A} &= I + i\theta A - \frac{\theta^2 I}{2!} - i\frac{\theta^3 A}{3!} + \frac{\theta^4 I}{4!} + i\frac{\theta^5 A}{5!} + \ldots \\ &= \left(I - \frac{\theta^2 I}{2!} + \frac{\theta^4 I}{4!} - \ldots \right) + iA\left(\theta - \frac{\theta^3}{3!} + \frac{\theta^5}{5!} - \ldots \right) \\ &= \cos\theta I + i \sin\theta A \end{aligned} \tag{2.61}$$

So depending on the fact that $X^2 = I$, $Y^2 = I$ and $Z^2 = I$, we define the rotation operators R_x, R_y, R_z as follows

$$R_x(\theta) \equiv e^{-i\theta X/2} = \cos\frac{\theta}{2} I - i \sin\frac{\theta}{2} X = \begin{bmatrix} \cos\frac{\theta}{2} & -i\sin\frac{\theta}{2} \\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{bmatrix}, \tag{2.62}$$

$$R_y(\theta) \equiv e^{-i\theta Y/2} = \cos\frac{\theta}{2} I - i\sin\frac{\theta}{2} Y = \begin{bmatrix} \cos\frac{\theta}{2} & -i\sin\frac{\theta}{2} \\ i\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{bmatrix}, \tag{2.63}$$

$$R_z(\theta) \equiv e^{-i\theta Z/2} = \cos\frac{\theta}{2} I - i\sin\frac{\theta}{2} Z = \begin{bmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{bmatrix}, \tag{2.64}$$

The operator $R_n(\theta)$ is defined to be the one which rotates the Bloch vector about the axis n and through the angle θ. Let us consider $R_x(\pi)$ as an example. It turns out that the Pauli X operator is equivalent to a rotation of π radians about the *x*-axis.

$$R_x(\pi) = \begin{bmatrix} \cos\frac{\pi}{2} & -i\sin\frac{\pi}{2} \\ -i\sin\frac{\pi}{2} & \cos\frac{\pi}{2} \end{bmatrix} = \begin{bmatrix} 0 & -i \\ -i & 0 \end{bmatrix} = -iX \tag{2.65}$$

The rotation operators are important because if we had them for arbitrary real values of θ then they would provide a basis on which we could build an arbitrary single qubit operator. There are several theorems stating this fact. One of them is the *Z*-*Y* decomposition theorem, which states that an arbitrary unitary operator *U* on a single qubit can be represented as

$$U = e^{i\alpha} R_z(\beta) R_y(\gamma) R_z(\delta), \tag{2.66}$$

where $\alpha, \beta, \gamma, \delta$ are real numbers [4]. In practice, we can only have the approximations of some unitary operators in this way, since we are restricted to the rational approximations of the real values. Here are two other one-bit unitary operators that will be of use in the following sections; the phase gate U_{phase} and the $\pi/8$ gate $U_{\pi/8}$.

$$U_{phase} = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}, \quad U_{\pi/8} = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix} \tag{2.67}$$

2.4.2. Multiple Qubit Quantum Gates

A quantum gate with *n*-bit input is a unitary mapping defined on the 2^n dimensional Hilbert space. The action of such a gate can be defined by its effect on the 2^n members of the computational basis. A unitary matrix is often used to model this action. Figure 2.5 contains the circuit representation for an *n*-bit operator *U* acting on a register $|\psi_1\psi_2..\psi_n\rangle$ of *n* qubits.

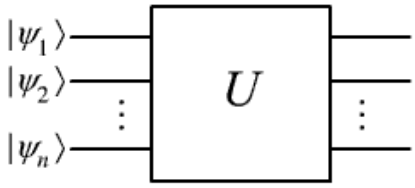

Figure 2.5. *n* bit operator U acting on the register $|\psi_1\psi_2..\psi_n\rangle$

An operator acting on a multi-particle system is called decomposable if its action on its input can be represented as the product of its action on the individual components of the input. Let U be a decomposable operator acting on a two qubit register such as $\psi = |\psi_1\psi_2\rangle$. Then there exist the single qubit operators U_1 and U_2 which satisfy the following equalities

$$U|\psi\rangle = U_1|\psi_1\rangle \otimes U_2|\psi_2\rangle \tag{2.68}$$

$$U = U_1 \otimes U_2 \tag{2.69}$$

Then the two circuits shown in Figure 2.6 are equivalent and they both represent the same computation, $U|\psi\rangle = U_1|\psi_1\rangle \otimes U_2|\psi_2\rangle$.

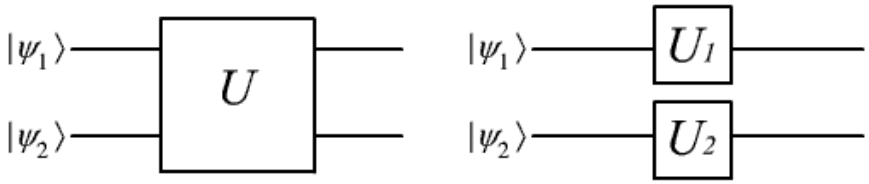

Figure 2.6. The alternative circuit representations for $U|\psi\rangle = U_1|\psi_1\rangle \otimes U_2|\psi_2\rangle$

As another example, let V be a two bit operator which leaves the first bit unchanged and acts as a NOT (Pauli X) gate on the second bit. (Note that applying identity operator to a bit is equivalent to leaving it unchanged.) Then we have

$$V = I \otimes X = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \otimes \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}. \tag{2.70}$$

The effect of V on the computational basis is defined by

$$V|00\rangle = |01\rangle, \quad V|01\rangle = |00\rangle, \quad V|10\rangle = |11\rangle, \quad V|11\rangle = |10\rangle. \tag{2.71}$$

Multi-bit Hadamard gates are commonly used in the quantum computation literature. In general, an n-bit Hadamard gate is given by $H_n = \bigotimes_1^n H$ where H denotes the familiar one-bit Hadamard gate. When applied to a base state $|k\rangle$, H_n generates an equal superposition of the 2^n members of the computational basis with varying relative phases depending on the initial state as shown in (2.72) where $k \cdot j = \sum_{i=0}^{n-1} k_i \cdot j_i$.

$$H_n|k\rangle = \frac{1}{\sqrt{2^n}} \sum_{j=0}^{2^n-1} (-1)^{k \cdot j} |j\rangle \tag{2.72}$$

Most of the interesting multi-bit quantum gates, including the CNOT and the Toffoli gates, are not decomposable. Their effect on a component of the input cannot be defined without reference to the other parts of the input. Let us examine some of these gates.

The controlled-NOT or CNOT gate is a two qubit gate where the input register can be labeled as $|control\rangle|target\rangle$. If we consider the computational basis, the effect of the CNOT

gate is like a conditional statement: if the control bit is in state $|1\rangle$ then apply NOT gate to the target bit. Hence it performs the transformation $|control\rangle|target\rangle \rightarrow |control\rangle|target \oplus control\rangle$ where $\oplus$ is the logical XOR. The circuit representation for CNOT is shown in Figure 2.7.

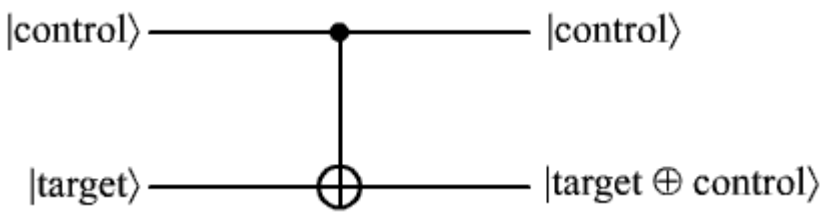

Figure 2.7. The circuit representation for the CNOT gate

The effect of CNOT on the computational basis is given by the following equations.

$$CNOT|00\rangle = |00\rangle, \quad CNOT|01\rangle = |01\rangle, \quad CNOT|10\rangle = |11\rangle, \quad CNOT|11\rangle = |10\rangle \tag{2.73}$$

Then, by equation (2.43), the matrix representation of CNOT can be calculated as

$$CNOT = |00\rangle\langle 00| + |01\rangle\langle 01| + |11\rangle\langle 10| + |10\rangle\langle 11| = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{2.74}$$

For an arbitrary single-bit quantum gate *U,* we can define a two-bit *controlled-U* operation. Then in terms of the computational basis, the action is defined by the conditional statement: if the control bit is in state $|1\rangle$ then apply *U* to the target bit. It is also possible to define the *zero-controlled-U* operation in which case the action on the computational basis is defined by the conditional statement: if the control bit is in state $|0\rangle$ then apply *U* to the target bit. The circuit representations for *controlled-U* and *zero-controlled-U* operations are given in Figure 2.8.

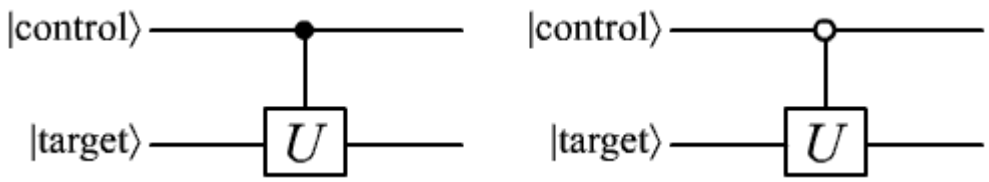

Figure 2.8. Circuit representations for controlled-U (on the left) and zero-controlled-U

Let us now consider the two-bit swap (S) operator whose effect on the computational basis is simply swapping the states of the input bits. Hence, we have

$$S|00\rangle = |00\rangle, \quad S|01\rangle = |10\rangle, \quad S|10\rangle = |01\rangle, \quad S|11\rangle = |11\rangle \tag{2.75}$$

Then the unitary matrix for S is given by

$$S = |00\rangle\langle 00| + |01\rangle\langle 10| + |10\rangle\langle 01| + |11\rangle\langle 11| = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.76}$$

Another interesting multi-bit quantum gate is the Toffoli gate which acts on a register of three qubits $|control\ 1\rangle|control\ 2\rangle|target\rangle$. The first two bits are the control bits and the last bit is the target bit. The effect of the Toffoli gate on the computational basis is defined by the conditional statement: If both of the control bits are in state $|1\rangle$ then apply *NOT* to the target bit. The circuit representation for the Toffoli gate is given in Figure 2.9.

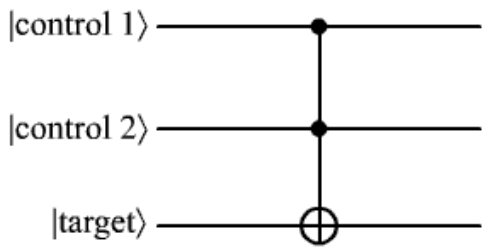

Figure 2.9. The circuit representation for the Toffoli gate

Note that it is possible to define operators with arbitrary number of control bits and arbitrary combinations of zero and one controls. In particular, the Toffoli gate is nothing but a double controlled not gate. The Fredkin gate on the other hand implements controlled swap operation and hence it is a three-qubit operator with single control bit. Below are the matrix representations for the Toffoli (U_T) and Fredkin (U_F)gates.

$$U_T = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}, \quad U_F = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.77}$$

2.4.3. The Reversibility of Quantum Gates

A computational process is said to be reversible if its outputs can be used to uniquely determine its inputs. Such a process is one where there is no information loss. We are interested in reversible computation because the quantum gates have to be reversible as a result of unitarity and hence only those computational processes which can be implemented reversibly can be applied to quantum computers. In general, a quantum algorithm can be conceived as a unitary transformation U taking the state $|input\rangle$ to the state $|output\rangle$. Then we also have $U^\dagger$ which satisfies $U^\dagger|output\rangle = |input\rangle$.

The initial interest in reversible computation has much different roots. Most probably, it has developed in order to find a way of limiting the energy consumption of the computational processes since the laws of thermodynamics indicates that erasing every bit of information requires some energy dissipation (Landauer's principle) and the standard computational processes are in no way reversible. Note that none of the classical gates AND, OR, NAND are reversible and hence each of them requires some energy to be performed.

No matter what motivates it, there is a literature about classical reversible computation which puts it forward that every classical computational process can be implemented in a reversible manner. A simple proof of this fact can be built on the idea that any irreversible operation can be made reversible by additional input and output ports. In general, an irreversible operation with *n* inputs and *m* outputs can be implemented reversibly with $n+m$ inputs and outputs. (Consider the classical AND operation which has two inputs *x*, *y* and one output, $x \wedge y$. This gate is surely not reversible. The reversible AND operation with three inputs and three outputs can be implemented by a gate which performs $x, y, z \to x, y, z \oplus x \wedge y$ with input *x*, *y,* 0.) Then any classical circuit can be transformed to a reversible one by just replacing the irreversible gates with their reversible counterparts. (There is, however, one extra issue here. There may be supplementary bits used in the computation which do not appear in the inputs and outputs. We name them ancilla bits. A reversible computational process needs to restore the initial values of these bits after the necessary computations are made. This task is known as uncomputation.) A more straight way to show the same result, is of course demonstrating the universality of 3-input nonlinear reversible gates (such as Toffoli and the Fredkin gates) for classical computation. We will skip this until the next section.

Universality of reversible computation is an important result for quantum computation because any classical subroutine in a quantum algorithm has to be implemented reversibly. For a more detailed discussion the topic, please refer to [4, 9, 10].

2.4.4. The Universality of Quantum Gates

In classical computation, a set of gates or operators is said to be universal if its members are sufficient to compute a complete set of the classical boolean functions and thus to construct general combinatory circuits. If a computational device supports a universal set of operations, than it can be said to support a full range of classical operations. The sets {AND, NOT}, {OR, NOT} are among the well known examples of the universal sets of operations for classical computation.

Let us consider the Toffoli gate that was introduced in Section 2.4.2. The lines below show that the operations in the universal set {AND, NOT} can be implemented with use of a Toffoli gate.

$$U_T\left|1,1,x\right\rangle = \left|1,1,\sim x\right\rangle \tag{2.78}$$

$$U_T\left|x,y,0\right\rangle = \left|x,y,x \wedge y\right\rangle \tag{2.79}$$

Note that (2.78) and (2.79) demonstrate the universality of Toffoli gate for classical computation. (Hence, they also construct a proof for the universality of reversible computation.) Then, it can be stated that a quantum computer which supports the Toffoli gate can compute the complete set of the classical boolean functions and hence it can implement any classical algorithm.

On the other hand, a set of quantum gates is said to be universal if its members are sufficient to construct quantum circuits that can approximate any unitary operation to arbitrary accuracy. There are several important results related to the universality of quantum gates. Here we will briefly mention some of these without dealing with how they are derived. For a detailed discussion of the topic, see [4, 11].

The first result to be mentioned here states that any unitary matrix can be represented as a product of two-level unitary matrices where a two level unitary matrix is one which acts non-trivially only on two or fewer vector components. (At most two of the rows and at most two of the columns differ from those for the identity matrix of the same size.) Then two level unitary matrices are said to construct a universal set of operations for quantum computation.

The set of single qubit gates plus the CNOT operation can be used to implement arbitrary two level unitary matrices. Hence, single qubit gates plus the CNOT operation is another universal set for quantum computation. This set is more restricted but it is still infinite.

A finite set of gates cannot implement arbitrary unitary gates exactly, however, there are such sets that can be used to approximate any unitary operation to arbitrary accuracy. Hadamard, phase, CNOT and $\pi/8$ gates construct a universal set in this sense. Another set with the same property contains the Hadamard, phase, CNOT and Toffoli gates. It should also be stated here that not all of the arbitrary unitary gates can be approximated by circuits of polynomial size.

2.5. The Quantum No-Cloning Theorem

We have seen that the state of a quantum system evolves only according to unitary transformations. It is the linear nature of these unitary transformations out of which, one of the most interesting results of quantum mechanics is immediately derived. This result is known as the no-cloning theorem and is of extreme value for quantum cryptography and quantum error correction.

Consider a quantum system whose state lies in *n*-dimensional Hilbert space H_n, where $n > 1$. Then the no-cloning theorem simply says that an arbitrary state of such a system cannot be copied.

The theorem is derived as easily as it is stated. Let us suppose that the cloning mentioned above was possible. Then there exists a unitary operator *U* which acts on two registers of same size and whose action is just copying the state of the first register into the second. Suppose $|\psi\rangle$ and $|\varphi\rangle$ are two distinct states valid for the first register. Then we should have

$$U|\psi\rangle|0\rangle = |\psi\rangle|\psi\rangle \tag{2.80}$$

$$U|\varphi\rangle|0\rangle = |\varphi\rangle|\varphi\rangle \tag{2.81}$$

Let us consider the inner products side by side

$$\langle 0|\langle\varphi|U^{\dagger}U|\psi\rangle|0\rangle = |\varphi\rangle|\varphi\rangle\langle\psi|\langle\psi| \tag{2.82}$$

Then,

$$\langle\varphi|\psi\rangle = \left(\langle\varphi|\psi\rangle\right)^2 \tag{2.83}$$

Then it is either the case that $\langle\varphi|\psi\rangle = 0$ or $\langle\varphi|\psi\rangle = 1$. $\langle\varphi|\psi\rangle = 1$ can hold only if $|\psi\rangle = |\varphi\rangle$, since $|\psi\rangle$ and $|\varphi\rangle$ are assumed to be distinct it should be the case that $\langle\varphi|\psi\rangle = 0$. This holds only if $|\psi\rangle$ and $|\varphi\rangle$ are orthogonal. Then two distinct arbitrary states like $|\psi\rangle$ and $|\varphi\rangle$ can be copied by the same unitary operator only if they are orthogonal, which means that for $n > 1$, no unitary operator can copy arbitrary states in H_n. Note that there exists only one distinct state denoted by the unit ket vectors in H_1. Hence, the quantum no cloning theorem excludes the systems whose state lies in one dimensional Hilbert space.

In Section 2.4, we had stated that a quantum computer which supports the Toffoli gate can implement any classical algorithm. Now, is it not contradictory that quantum computers cannot make copies of the arbitrary states of a register while this is just an ordinary task for classical computers? The answer is no, since the task of copying the quantum mechanical information is not the same as copying the classical information. When implementing a classical algorithm with a quantum computer, it is only the base states of registers to be copied and not the superpositions of them. And since the base states are orthogonal to each other, there are unitary operations which achieve this task. The CNOT gate for example can copy the state of a one bit register which is known to be in a base state $|0\rangle$ or $|1\rangle$.

$$CNOT|0,0\rangle = |0,0\oplus 0\rangle = |0,0\rangle, \tag{2.84}$$

$$CNOT|1,0\rangle = |1,0\oplus 1\rangle = |1,1\rangle\ . \tag{2.85}$$

It should also be stated that imperfect copies which are good for some measure of interest can be built. For information about quantum copying in this sense, please see [5].

2.6. The Measurement

In the previous sections, it was stated that the state of a closed quantum system evolves only according to unitary transformations, however it was said, when such a system is measured, the system no longer remains closed so its state is governed by different rules. In this section, we will briefly see what these rules are. We will introduce the measurement operators and this will help us to see with which probabilities the different outcomes of a measurement occur and what happens to the state of a measured system. This is of course what meets the practical computational needs. [5] presents useful sketches of various interpretations of quantum measurements and for a quantum mechanical point of view see [12].

A collection $\{M_i\}$ of measurement operators, one for each possible outcome, describes the nature of a quantum measurement. Each of these operators acts on the state space of the system to be measured and as a collection they have to satisfy the completeness condition given below.

$$\sum_i M_i^\dagger M_i = I \tag{2.86}$$

If $|\psi\rangle$ is the state of the system to be measured, then the probability that the outcome *i* occurs is given by

$$p(i) = \langle \psi | M_i^\dagger M_i | \psi \rangle \tag{2.87}$$

These probabilities sum up to 1 by the linearity condition:

$$\sum_i p(i) = \sum_i \langle\psi|M_i^\dagger M_i|\psi\rangle = \langle\psi|\left(\sum_i M_i^\dagger M_i\right)|\psi\rangle = \langle\psi|\psi\rangle = 1 \tag{2.88}$$

The state of the system after a measurement where the *i*'th outcome is observed is

$$\frac{M_i|\psi\rangle}{\sqrt{\langle\psi|M_i^\dagger M_i|\psi\rangle}}. \tag{2.89}$$

It must be clear that the completeness relation does not define a unique set of measurement operators. The set to be used should be selected according to the type of information expected from the results of the measurement.

In particular if we are to measure a qubit in the computational basis, then there are two possible outcomes, $|0\rangle$ and $|1\rangle$. The operators defining this measurement are $M_0 = |0\rangle\langle 0|$ and $M_1 = |1\rangle\langle 1|$. (Note that these operators satisfy the completeness condition.) Then if the qubit is in state $|\psi\rangle = a|0\rangle + b|1\rangle$ before the measurement, the probabilities for observing $|0\rangle$ and $|1\rangle$ are calculated as follows.

$$p(0) = \langle\psi|M_0^\dagger M_0|\psi\rangle = \begin{bmatrix} a & b \end{bmatrix}\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} a \\ b \end{bmatrix} = |a|^2, \tag{2.90}$$

$$p(1) = \langle\psi|M_0^\dagger M_0|\psi\rangle = \begin{bmatrix} a & b \end{bmatrix}\begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} a \\ b \end{bmatrix} = |b|^2. \tag{2.91}$$

If $|0\rangle$ is observed, the state of the qubit collapses to $|0\rangle$,

$$\frac{M_0|\psi\rangle}{\sqrt{\langle\psi|M_0^\dagger M_0|\psi\rangle}} = \frac{\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} a \\ b \end{bmatrix}}{\sqrt{a^2}} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} = |0\rangle \tag{2.92}$$

and if $|1\rangle$ is observed, the state of the qubit collapses to $|1\rangle$.

$$\frac{M_1|\psi\rangle}{\sqrt{\langle\psi|M_1^\dagger M_1|\psi\rangle}} = \frac{\begin{bmatrix}0 & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}a\\ b\end{bmatrix}}{\sqrt{b^2}} = \begin{bmatrix}0\\ 1\end{bmatrix} = |1\rangle \tag{2.93}$$

The same reasoning explains how we can make measurements of the state of a qubit in any orthogonal basis. If, for example, we want to measure the state of a qubit in the Hadamard basis, then we choose $M_+ = |+\rangle\langle+|$ and $M_- = |-\rangle\langle-|$ as the measurement operations. These measurements are defined by orthogonal projection operators and hence they are called projective (or Von Neumann) measurements. For computational concerns, most of the interest is concentrated in projective measurements, so the measurements which are not projective are not considered here. See, however [4] for a detailed discussion of the topic.

2.7. Entanglement

We had seen that the state space H_s of a composite system is defined by the tensor product $\bigotimes_i H_i$ of the state spaces H_i for the individual components of the system. The state ψ_s of the system is said to be decomposable if it can be represented as a product $\bigotimes_i \psi_i$ of the members ψ_i of the state spaces H_i. Notice however that not all members of H_s are decomposable. Those states which are not decomposable are called the entangled states. As one of the simplest but the most famous example to the entangled states, we can consider the following two qubit register sate which is a member of the family of states known as the Bell states or the EPR (Einstein, Podolsky, Rosen) pairs.

$$|\psi\rangle = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\ 0\\ 0\\ 1\end{bmatrix} \tag{2.94}$$

To show that $|\psi\rangle$ is entangled, we simply need to show that it cannot be decomposed into two single qubit states. Let us suppose it was decomposable and let $|\psi\rangle = |\psi_1\rangle|\psi_2\rangle$ where $|\psi_1\rangle = a|0\rangle + b|1\rangle$ and $|\psi_2\rangle = c|0\rangle + d|1\rangle$. Then we have

$$|\psi\rangle = \begin{bmatrix} a \\ b \end{bmatrix} \otimes \begin{bmatrix} c \\ d \end{bmatrix} = \begin{bmatrix} ac \\ ad \\ bc \\ bd \end{bmatrix} \tag{2.95}$$

But then we have $ac = bd = 1/\sqrt{2}$ and $ad = bc = 0$ which is a contradiction since we have $ac = 1, ad = 0, bc = 0, bd = 1$ which implies $(ac)(bd) \neq (ad)(bc)$. Therefore, we can conclude that there are no such single qubit states $|\psi_1\rangle$ and $|\psi_2\rangle$, for which it holds that $|\psi\rangle = |\psi_1\rangle|\psi_2\rangle$. So $|\psi\rangle$ is not decomposable and hence it is an entangled state.

The circuit shown in the Figure 2.10 generates the four Bell states $|\beta_{00}\rangle$, $|\beta_{01}\rangle$, $|\beta_{10}\rangle$ and $|\beta_{11}\rangle$ when it is initialized with the four members of the computational basis $|00\rangle$, $|01\rangle$, $|10\rangle$ and $|11\rangle$ respectively. Note that $|\psi\rangle$ corresponds to the input $|00\rangle$ hence it is equivalent to $|\beta_{00}\rangle$.

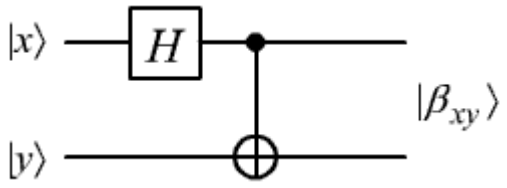

Figure 2.10. The circuit that generates the Bell states when initialized in the basis states

Then the four Bell states are

$$\beta_{00} = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\0\\0\\1\end{bmatrix},\ \beta_{01} = \frac{1}{\sqrt{2}}\begin{bmatrix}0\\1\\1\\0\end{bmatrix},\ \beta_{10} = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\0\\0\\-1\end{bmatrix},\ \beta_{11} = \frac{1}{\sqrt{2}}\begin{bmatrix}0\\1\\-1\\0\end{bmatrix}. \tag{2.96}$$

All of the above states are entangled and they are the origin of the many interesting ideas and discussions such as quantum teleportation and super-dense coding.

2.8. Quantum Computational Complexity

In order to develop an understanding of the power of quantum computation, there is the need for a theory of quantum computational complexity which defines complexity classes for quantum computation and relates them with each other as well as their classical counterparts. Although the theory is relatively new, several important results have already been established. Here we will introduce two of the most important quantum complexity classes and mention some of the important results in the field without giving details of their derivation.

Most of the researchers working on quantum computational complexity use a quantum analogue of the Turing Machine as a universal model of quantum computation after the introduction of the first universal quantum Turing machine (QTM) by Deutsch [13]. The QTM model proposed by Bernstein and Vazirani was the first which can simulate any other QTM with only polynomial slowdown [14]. Definition of several such models can also be found in [9, 15]. The quantum circuit model can be shown to be equivalent to QTM models in terms of the computational power [16]. Hence in the following parts we will continue to refer quantum circuits rather than QTMs. (A third equivalent model is that of quantum computation by adiabatic evolution, see [17].)

The most trivial quantum complexity class is EQP (exact quantum polynomial time), which contains those problems that can be solved to certainty by quantum circuits of polynomial size in the worst case. As any classical circuit can be implemented as a quantum

circuit with at most a polynomial overhead, we can safely state that EQP is at least as large as the classical complexity class P.

The most important quantum computational complexity class is BQP (bounded-error quantum polynomial time). A language $L \subseteq \Sigma^*$ (Σ^* denotes the set of strings that can be written with the symbols in the alphabet Σ.) is said to be in BQP if and only if there is a quantum circuit C of polynomial size which decides L in the sense that, for any $x \in \Sigma^*$, C accepts x with probability at least $2/3$ if $x \in L$ and it rejects x with probability at least $2/3$ if $x \notin L$. In practice this amounts to running the circuit C on the input x. After the computation the decision, whether $x \in L$ or not, is encoded into one of the output qubits: $a|0\rangle + b|1\rangle$. We measure this bit and take, for example, $|0\rangle$ as accept and $|1\rangle$ as reject. Then every time C is run on $x \in L$, it should be the case that $|a|^2 \geq 2/3$ and every time C is run on $x \notin L$, it should be the case that $|b|^2 \geq 2/3$. The probability of error can be reduced to an arbitrarily small value ε, by running the circuit $O(log\, 1/\varepsilon)$ times and taking the majority of the results. Hence, the problems in BQP are efficiently solvable on a quantum computer. By its definition BQP is at least as large as EQP. Then we have the following chain of inclusion relations.

$$\mathrm{P} \subseteq \mathrm{EQP} \subseteq \mathrm{BQP} \tag{2.97}$$

A very important result states that BQP is contained in the classical complexity class PSPACE [18]. Since it is straightforward that quantum computers are at least as powerful as classical computers, we also know that BPP (bounded error probabilistic polynomial time) is contained in BQP. Then we have another chain

$$BPP \subseteq BQP \subseteq PSPACE \tag{2.98}$$

It is not yet known whether any of these inclusion relations are strict. Hence if one could show that BQP is strictly larger than BPP (or equivalently, that quantum computers can

effectively solve more problems than classical computers) then this will also provide a very important result for classical complexity theory saying that PSPACE is strictly larger than BPP and hence than P.

Another important class is BQNP, which is the quantum analogue of the classical complexity class NP. A language L is said to be in BQNP if and only if, there is a polynomial size quantum proof checker circuit *C.* For every $x \in L$, there is a proof string *y* such that the proof checker accepts the input *x,y* with probability at least $2/3$ and for every $x \notin L$ and for every *y,* the proof checker rejects the input *x,y* with probability at least $2/3$. A quantum analogue of the Cook-Levin theorem has been demonstrated by Kitaev. It states that the problem QSAT (the quantum analogue of the 3-SAT problem) is complete for the BQNP class [19].

When we assume there are no limits on the computational resources and that we cannot implement arbitrary complex numbers as amplitudes, the set of problems that can be solved on a quantum computer is exactly the same as that for a classical computer. So the quantum computers do not violate the Church-Turing thesis stating that any function that can be computed by any means can also be computed by a Turing machine. However when the efficiency is taken into the consideration quantum computation constructs a though challenge for the so called strong version of the Church-Turing thesis stating that any computable function can be computed on a Turing machine with at most a polynomial increase in the running time.

There are many other important ideas and results in the quantum computational complexity domain. It is out of the scope of this work to mention all of them, however the reader is encouraged to see [15, 20] and chapter 5 of [9] for a more detailed discussion of the topic.

2.9. Quantum Algorithms

A quantum computer is not necessarily faster than a classical computer when performing an ordinary task such as addition or multiplication. The superiority of quantum computers over the classical ones depends on mechanisms such as superposition, interference and entanglement. This superiority can be demonstrated by quantum algorithms which outperform their classical counterparts. This is why most of the literature about quantum algorithms has focused on these algorithms. So far, a few techniques are known to generate such algorithms and yet it is not known whether there are other such techniques to be discovered.

We can categorize the quantum algorithms with respect to the nature of the problem they are devised for and the qualitative measure of the performance gain they provide against the classical algorithms. The first division is roughly between the conventional problems and black box problems. A black box (or an oracle), in computer science, can be taken as anything that computes and returns a function $f(x)$ of any legal input x. A black box problem is one, which differs from conventional problems by providing access to a black box in its definition, with no care, and knowledge of what happens in it. Although they are artificial problems they are of theoretical importance in computer science. The second division is between the algorithms, which provide sub-exponential speedup over the classical algorithms and those, which provide exponential speedup. Although a sub-exponential speedup is still of great importance, it is only the exponential speedup, that demonstrates the full power of quantum computation.

The best and probably the most famous example of sub-exponential speedup for a black box problem is achieved by Grover's algorithm for unstructured search [1] which depends on the amplitude amplification method [21]. This algorithm is applied to the cases where a linear search seems to be the most rational classical choice and provides quadratic speedup over the best classical algorithm for the same task. Grover's algorithm can be used to speed up the solutions of NP complete problems, however neither this algorithm nor any other algorithm in this category, provide meaningful results for the conventional complexity classifications.

One of the first problems for which quantum computers were shown to outperform the classical computers is a black box problem (known as Deutsch's problem) about distinguishing between constant and balanced functions [13]. It is for a generalization of this problem that Deutsch and Jozsa suggested their famous algorithm which depends on the superposition and interference principles and can easily be shown to be exponentially faster than any deterministic classical algorithm for the same task [22]. However this algorithm is not superior to a probabilistic classical algorithm, and hence its importance is limited.

Another important algorithm that provides exponential speedup for a black box problem is Simon's algorithm, which is devised for the problem of finding the period of a 2 to 1 function [23]. This problem can be formulated as a hidden subgroup problem and Simon's work on period finding can be said to provide the base on which Shor's factorization algorithm is built. Unlike the Deutsch-Jozsa algorithm, Simon's algorithm is superior to all of its classical counterparts, deterministic or probabilistic. Hence, in [24], Shor states that Simon's algorithm provides "a moderately convincing argument that BQP is strictly larger than BPP, although it is not a rigorous proof."

Probably the most celebrated achievement of quantum computation is the discovery of Shor’s algorithm which provides exponential speedup (over the best known classical algorithm) for a conventional problem, factorization [2]. The factorization problem is not proven but highly suspected to be classically hard that even probabilistic classical algorithms may not be able to provide polynomial solutions for it. Shor's algorithm, introduced in 1994, shows that a polynomially large quantum circuit can solve the factorization problem depending on the phase estimation technique which can be applied efficiently, with use of a quantum implementation of the Fourier transformation method. This remains as one of the most important results in the field of quantum computation, not only because the factorization problem has many applications and hence is extensively studied in the classical domain, but also because the methods used in this algorithm can be generalized to provide efficient solutions for the samples of the other hidden subgroup problems [25, 26]. However, since the complexity of factorization is not known, this algorithm does not provide means to understand the relation between BPP and BQP.

It is out of the scope of this work to provide detailed discussions about all of these algorithms. In sections 2.9.1, 2.9.2 and 2.9.3 below, we will provide sketches of Simon's, Grover's and Shor's algorithms to give an idea of the general characteristics of quantum algorithms. The reader, however, is referred to [27] for a detailed analysis of each of the algorithms mentioned above.

2.9.1. Simon's Algorithm for Period Finding

Problem: Given a black box U_f which computes a function $f:\{0,1\}^n \to \{0,1\}^m$ ($m \geq n$) that is known either to be one to one or to satisfy (2.99) for a non-trivial *s*, where $\oplus$ denotes bitwise addition modulo 2.

$$(x \neq y) \wedge (f(x) = f(y)) \leftrightarrow y = x \oplus s \tag{2.99}$$

The problem is to determine if *f* is one to one, if it is not then to find *s*. (We denote the functionality of U_f as a unitary transformation: $U_f(|x\rangle \otimes |d\rangle) = |x\rangle \otimes |d \oplus f(x)\rangle$)

Algorithm: Simon's algorithm consists of the following steps.

Step 0: Initialize two quantum registers of n and m qubits in the states $\bigotimes_0^{n-1}|0\rangle$ *and* $\bigotimes_0^{m-1}|0\rangle$.

Step 1: Apply n-bit Hadamard gate H_n *to the first register of n bits.* The overall state will be as shown in (2.100). Note that this step puts the first register into an equal superposition of the 2^n principal states.

$$\left(H_n \bigotimes_0^{n-1}|0\rangle\right) \otimes \left(\bigotimes_0^{m-1}|0\rangle\right) = \left(\frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1}|k\rangle\right) \otimes \left(\bigotimes_0^{m-1}|0\rangle\right). \tag{2.100}$$

Step 2: *Query the black box for the state prepared in Step 1.* Then the next state is

$$U_f\left(\left(\frac{1}{\sqrt{2^n}}\sum_{k=0}^{2^n-1}|k\rangle\right)\otimes\left(\bigotimes_{0}^{m-1}|0\rangle\right)\right)=\frac{1}{\sqrt{2^n}}\sum_{k=0}^{2^n-1}|k\rangle\otimes|f(k)\rangle. \tag{2.101}$$

Step 3: Apply n-bit Hadamard gate H_n to the first register again. The new state is

$$(H_n\otimes I_m)\left(\frac{1}{\sqrt{2^n}}\sum_{k=0}^{2^n-1}|k\rangle\otimes|f(k)\rangle\right)=\frac{1}{\sqrt{2^n}}\sum_{k=0}^{2^n-1}\left(\frac{1}{\sqrt{2^n}}\sum_{j=0}^{2^n-1}(-1)^{j\cdot k}|j\rangle\otimes|f(k)\rangle\right). \tag{2.102}$$

Now, if *f* is one to one then both the domain and range (mirror of the domain under *f*) of *f* has the same cardinality 2^n. The state shown in (2.102) is a superposition of the members of the Cartesian product of the domain and range of *f*. Therefore, $2^n 2^n = 2^{2n}$ states of the form $|j\rangle\otimes|f(k)\rangle$ are superposed, each with an amplitude equal to either $\frac{1}{2^n}$ or $-\frac{1}{2^n}$. Then, if the state of the first register is measured after *Step 3*, the probabilities for observing each of the 2^n base states are equal and given by $2^n\left|\pm\frac{1}{2^n}\right|^2=\frac{1}{2^n}$. Hence the outcome of such a measurement would be a random value between 0 and 2^n-1.

If *f* is not one to one, it satisfies (2.99). Then each state of the form $|j\rangle\otimes|f(k)\rangle$ has the amplitude given by

$$\frac{1}{2^n}\left((-1)^{j\cdot k}+(-1)^{j\cdot(k\oplus s)}\right)=\frac{1}{2^n}\left((-1)^{j\cdot k}+(-1)^{(j\cdot k)\oplus(j\cdot s)}\right). \tag{2.103}$$

If $j\cdot s\neq 0$ then the amplitude for $|j\rangle\otimes|f(k)\rangle$ becomes zero. So if *f* is not one to one, then measuring the state of the first register after *Step 3,* returns a value of *j* such that $j\cdot s=0$.

Step 4: Measure the state t of the first register.

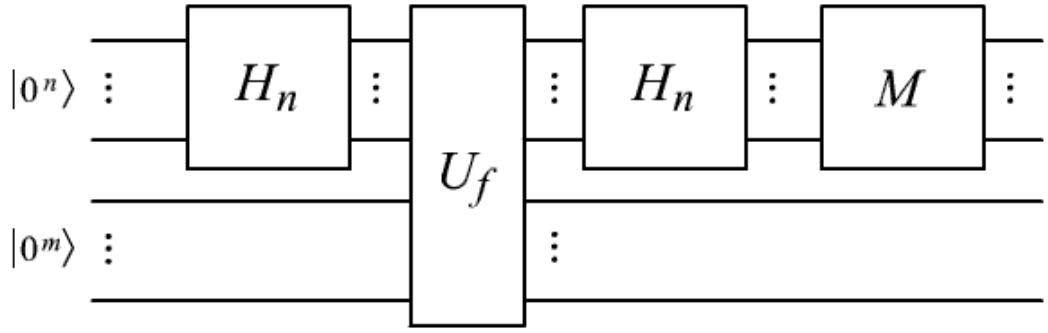

Figure 2.11. The circuit representation for the iterative steps of Simon's algorithm

The circuit representation for steps 0 through 4 is shown in the Figure 2.11. If we run this circuit succesively, until we observe n linearly independent values of t, we get a system of equations: $t_1 \cdot s = 0, t_2 \cdot s = 0,.., t_n \cdot s = 0$. This is a system of n linear equations in n unknowns and it can be solved for the n bits of s classically in polynomial time. Once we get the value for s, we can query the black box for an arbitrary value v, between 0 and $2^n - 1$ to see if $f(v) = f(v \oplus s)$. If this is not the case we can conclude that f is one to one and otherwise it satisfies (2.99) for the value of s which has already been calculated.

The expected running time of the algorithm is O(n), however in the worst case it may take too long to observe n linearly independent values for t. A slightly modified version of the algorithm can get over this problem by removing the observed values of t from the initial superpositions of the following iterations [28].

The best classical algorithm for the same task would require exponentially many queries of the black box in terms of n. Therefore, Simon's algorithm is exponentially faster than the best possible classical algorithm for the same task. Moreover, it is the first algorithm that depends on the idea of realizing the periodic properties of a function in the relative phase factors of a quantum state and then transforming it into information by means of probability distribution of the observed states [27]. The ideas used in the period finding algorithm turned out to be useful for developing algorithms for many other problems.

2.9.2. Grover's Algorithm for Unstructured Search

Problem: Given a black box U_f for the function $f:\{0,1\}^n \to \{0,1\}$ such that there exists exactly one $x \in \{0,1\}^n$ satisfying $f(x)=1$. The problem is to find that *x*. (We denote the functionality of U_f as a unitary transformation: $U_f\left(|x\rangle \otimes |d\rangle\right) = |x\rangle \otimes |d \oplus f(x)\rangle$)

Algorithm: Grover's algorithm consists of the following steps.

Step 0: Initialize two quantum registers, α of n qubits and β of 1 qubit in the states $\alpha = \bigotimes_0^{n-1} |0\rangle$ *and* $\beta = |1\rangle$.

Step 1: Apply Hadamard gate H to all of the qubits in the registers α and β. The overall state will be as shown in (2.104).

$$\begin{aligned}\left(\bigotimes_0^{n-1} H|0\rangle\right) \otimes \left(H|1\rangle\right) &= \left(\frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} |k\rangle\right) \otimes \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) \\ &= \frac{1}{\sqrt{2^{n+1}}} \sum_{k=0}^{2^n-1} |k\rangle\left(|0\rangle - |1\rangle\right) \\ &= \frac{1}{\sqrt{2^{n+1}}}\left(\sum_{k=0}^{2^n-1} |k\rangle|0\rangle - \sum_{k=0}^{2^n-1} |k\rangle|1\rangle\right)\end{aligned} \tag{2.104}$$

Let X_S denote the set of all *n*-bit sequences *x* that satisfy $f(x)=1$ and X_T denote the set of all *n*-bit sequences *x* that satisfy $f(x)=0$. We have $X_S \cup X_T = \{0,1\}^n$ so that we can rewrite the state in (2.104) as

$$\frac{1}{\sqrt{2^{n+1}}}\left(\sum_{k=0}^{2^n-1} |k\rangle|0\rangle - \sum_{k=0}^{2^n-1} |k\rangle|1\rangle\right) = \frac{1}{\sqrt{2^{n+1}}}\left(\sum_{x \in X_S} |x\rangle|0\rangle + \sum_{x \in X_T} |x\rangle|0\rangle - \sum_{x \in X_S} |x\rangle|1\rangle - \sum_{x \in X_T} |x\rangle|1\rangle\right). \tag{2.105}$$

Step 2: Apply U_f on the registers α and β. The new state will be

$$\begin{aligned}
&\frac{1}{\sqrt{2^{n+1}}}U_f\left(\sum_{x\in X_S}|x\rangle|0\rangle+\sum_{x\in X_T}|x\rangle|0\rangle-\sum_{x\in X_S}|x\rangle|1\rangle-\sum_{x\in X_T}|x\rangle|1\rangle\right)\\
&=\frac{1}{\sqrt{2^{n+1}}}\left(\sum_{x\in X_S}|x\rangle|0\oplus f(x)\rangle+\sum_{x\in X_T}|x\rangle|0\oplus f(x)\rangle-\sum_{x\in X_S}|x\rangle|1\oplus f(x)\rangle-\sum_{x\in X_T}|x\rangle|1\oplus f(x)\rangle\right). \qquad (2.106)\\
&=\frac{1}{\sqrt{2^{n+1}}}\left(\sum_{x\in X_S}|x\rangle|1\rangle+\sum_{x\in X_T}|x\rangle|0\rangle-\sum_{x\in X_S}|x\rangle|0\rangle-\sum_{x\in X_T}|x\rangle|1\rangle\right)
\end{aligned}$$

If we consider the individual states of the registers α and β, we can rewrite the overall state as follows

$$\alpha\otimes\beta=\frac{1}{\sqrt{2^n}}\left(\sum_{x\in X_T}|x\rangle-\sum_{x\in X_S}|x\rangle\right)\otimes\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right). \qquad (2.107)$$

Now it is seen in the first register that the amplitudes for those bit sequences *x* that satisfy $f(x)=1$ are negative while the amplitudes for those bit sequences *x* that satisfy $f(x)=0$ are positive. So the steps up to this point have created a difference between the states we want to distinguish. However at this point the difference is not informative since the difference in amplitudes is not reflected in the observation probabilities: any bit sequence has equal probability of being observed. Hence, there is the need to transform the difference in amplitudes into a difference in the observation probabilities. Since we want to learn which particular bit sequences *x* satisfy $f(x)=1$ we need to increase the probability of observing those with negative amplitudes. This is done by the so-called "inversion around average" operator (U_{IA}). It simply maps each amplitude a_i in the state vector to $2A-a_i$ where A is the average of all the amplitudes [5]. (Note that a_i and $2A-a_i$ are symmetric around A.) The 2^n by 2^n U_{IA} operator can be written as follows.

$$U_{IA} = \begin{bmatrix} \frac{2}{2^n} & \cdots & \frac{2}{2^n} \\ \vdots & \ddots & \vdots \\ \frac{2}{2^n} & \cdots & \frac{2}{2^n} \end{bmatrix} - I_{2^n} \tag{2.108}$$

Step 3: Apply U_{IA} on the register α.

It is known by the problem statement that X_S contains only one bit sequence and X_f contains $2^n - 1$ such bit sequences. Therefore, the average of the amplitudes in the state vector for the register α before *Step 3* is as follows, assuming large values for *n*.

$$A = \frac{\frac{(-1)}{\sqrt{2^n}} + \frac{(2^n - 1)}{\sqrt{2^n}}}{2^n} \approx \frac{1}{\sqrt{2^n}} \tag{2.109}$$

It can be stated that in *Step 3*, U_{IA} maps the amplitudes for the bit sequences in X_S to

$$2A - \frac{(-1)}{\sqrt{2^n}} \approx \frac{3}{\sqrt{2^n}}, \tag{2.110}$$

and it maps the amplitudes for the bit sequences in X_T to

$$2A - \frac{(1)}{\sqrt{2^n}} < \frac{1}{\sqrt{2^n}}. \tag{2.111}$$

Hence if a measurement is made probability of observing the bit sequence *x* that satisfies $f(x) = 1$, is increased by almost nine times (assuming large *n*) after *Step 3* while the probabilities for observing the other sequences are relatively decreased. *Step 2* and *Step 3* can be said to push the general state vector towards the solution of the problem by increasing the

amplitude of the solution state. (This is the technique known as amplitude amplification.) However, the probability of observing the solution state is still small to be useful. It can be increased more by further applications of *Step 2* and *Step 3* in a loop. (These steps are known as Grover iteration.) The amplification factor changes at each iteration as the individual amplitudes and their average change. It can be shown that $\frac{\pi}{4}\sqrt{2^n}$ iterations would bring the state vector to its optimal position and further iterations would destroy this optimality [29]. Therefore, the next step is a loop.

Step 4: Repeat Step2 and Step3 until the total iteration count is equal to the integer nearest to $\frac{\pi}{4}\sqrt{2^n}$.

If we measure the state of register α after $\left\lceil \frac{\pi}{4}\sqrt{2^n} \right\rceil$ Grover iterations, the probability of observing the bit sequence *x* for which $f(x)=1$ is greater than $1-1/2^n$. So we measure it.

Step 5: Measure the state of the first register α .

The circuit representation for steps 0 through 5 is shown in Figure 2.12. The probability of getting a wrong answer after running this circuit is less than $1/2^n$.

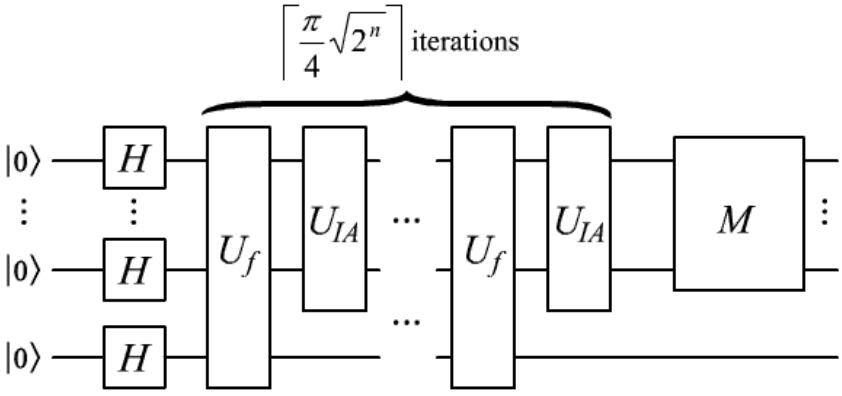

Figure 2.12. The circuit representation for Grover's algorithm

The same set of ideas can be applied to a more relaxed version of Grover's problem where it is allowed to have m different bit sequences of length n that satisfy $f(x)=1$. Assuming m is small with respect to 2^n, the problem can be solved (for one of the m solutions) by a modified version (which contains at most $\left\lceil \frac{\pi}{4}\sqrt{\frac{2^n}{m}} \right\rceil$ Grover iterations) of the circuit in Figure 2.12, with a probability of error at most $m/2^n$. If m is not known to be smaller than $2^n/2$, then it may be a good idea to enlarge the state space with the addition of 2^n more bit sequences, none of which is a solution.

The best classical algorithm for Grover's problem runs in $O(2^n)$ time since it needs to check every alternative bit sequence until it finds one which satisfies $f(x)=1$. Therefore Grover's search algorithm which runs in $O(\sqrt{2^n})$ time can be said to provide a quadratic speedup over the best classical algorithm for the same task. It was further shown that Grover's algorithm is optimal for the unstructured search problem [30].

2.9.3. Shor's Algorithm for Factorization

Given a composite integer N, find its prime factors. This is how the factorization problem, one of the classics in computer science, is stated. So simple in appearance, this problem is the subject of a broad range of studies and yet can be solved by the best known classical algorithm only in exponential time. Although there is no evidence, there is a strong belief that the factorization problem is classically hard. And this is why many security systems depend on the practical impossibility of factoring large integers with classical computers.

A quantum algorithm known with its inventor's name, Peter Shor, presents efficient means of solving the factorization problem. Shor's algorithm depends on some number theoretic observations and the idea of period finding first introduced with Simon's work. Here we discuss the main aspects of Shor's algorithm.

We begin with the number theoretic preliminaries. First we define the order of an integer a in the base of a positive integer N as the smallest positive integer r such that $a^r = 1 (\mathrm{mod}\, N)$. (Note that r is at the same time the period of the function $f_{a,N}(x) = a^x (\mathrm{mod}\, N)$.) If r is even then we can write

$$a^r - 1 = \left(a^{r/2} - 1\right)\left(a^{r/2} + 1\right) = 0 (\mathrm{mod}\, N). \tag{2.112}$$

We know $\left(a^{r/2} - 1\right) \neq 0 (\mathrm{mod}\, N)$ since r is the smallest positive integer such that $a^r = 1 (\mathrm{mod}\, N)$. If $\left(a^{r/2} + 1\right) \neq 0 (\mathrm{mod}\, N)$ is also true, then it should be the case that the prime factors of N are distributed into the prime factors of $\left(a^{r/2} - 1\right)$ and $\left(a^{r/2} + 1\right)$. Then we have

$$\gcd\left(N, \left(a^{r/2} - 1\right)\right) > 1, \tag{2.113}$$

$$\gcd\left(N, \left(a^{r/2} + 1\right)\right) > 1. \tag{2.114}$$

This is the first step in constructing Shor's algorithm. The second step depends on the observation that the period finding algorithm of Simon can be generalized to any periodic function, hence one can compute the order of an integer in any base efficiently on quantum computers while no efficient classical algorithm is known for this task. Hence, Shor's algorithm for factorization consists of the following steps and only *Step* 2 requires quantum computers to be efficiently implemented.

Step 1: Take a positive integer m at random. Compute $\gcd(N, m)$. *If* $\gcd(N, m) \neq 1$ then $\gcd(N, m)$ *is a non trivial factor of N, the algorithm terminates.*

Step 2: Compute the order, P, of m in the base N. If P is odd or $\left(m^{P/2} + 1\right) = 0 (\mathrm{mod}\, N)$ *then go to Step 1.*

Step 3: Compute $\gcd\left(N, \left(m^{P/2} - 1\right)\right)$, *which gives a non trivial factor of N.*

Step 1 and *Step 3* can be computed efficiently on a classical computer with use of the Euclidean algorithm for finding greatest common divisors [31]. *Step* 2 can be efficiently computed by a quantum computer. The probability that P is odd is given by $1/2^k$ and the probability that $\left(m^{P/2}+1\right)=0(\mathrm{mod}\,N)$ can occur is given by $1/2^{k-1}$, where k is the number of distinct prime factors of N.

Recall that finding the order of an integer a in the base N is equivalent to finding the period of the function $f_{a,N}(x)=a^x(\mathrm{mod}\,N)$. Let us now see how the period finding problem can be solved efficiently on a quantum computer. First we should introduce the quantum analogue of the Fourier transform operation. The quantum Fourier transformation operation U_{QFT} on a quantum register of L bits acts on the computational basis as described in (2.115).

$$U_{QFT}|k\rangle=\frac{1}{\sqrt{2^L}}\sum_{j=0}^{2^L-1}e^{(2\pi ikj)/2^L}|j\rangle \tag{2.115}$$

In [4, 5] circuit models are presented for implementing this unitary transformation on registers of arbitrary size. Now we define the period finding algorithm which consists of the following steps.

Input: A circuit component U_f for computing a periodic function f defined on the set of natural numbers.

Step 0: Prepare two quantum registers X and Y in the initial state $|X\rangle|Y\rangle=|0^{Lx}\rangle|0^{Ly}\rangle$ where the lengths Lx and Ly are chosen so that 2^{Lx} is greater than the anticipated value of the period and Ly is large enough that register Y can store vlues of function f.

Step 1: Apply U_{QFT} to the register X.

If we let $Q = 2^{Lx}$ the new state can be represented as in (2.116)

$$\left(U_{QFT}\left|0\right\rangle\right)\otimes\left|0\right\rangle = \left(\frac{1}{\sqrt{Q}}\sum_{j=0}^{Q-1} e^{(2\pi i 0 j)/Q}\left|j\right\rangle\right)\otimes\left|0\right\rangle = \frac{1}{\sqrt{Q}}\sum_{j=0}^{Q-1}\left|j\right\rangle\left|0\right\rangle \tag{2.116}$$

Step 2: Apply the transformation $\left|a\right\rangle\left|b\right\rangle \xrightarrow{U_f} \left|a\right\rangle\left|b\oplus f(a)\right\rangle$ *on the registers* $\left|X\right\rangle\left|Y\right\rangle$. Then the state of the two registers becomes

$$U_f\left(\frac{1}{\sqrt{Q}}\sum_{j=0}^{Q-1}\left|j\right\rangle\left|0\right\rangle\right) = \frac{1}{\sqrt{Q}}\sum_{j=0}^{Q-1}\left|j\right\rangle\left|f(j)\right\rangle. \tag{2.117}$$

Step 3: Apply U_{QFT} *to the register X.* The new state will be as follows.

$$\begin{aligned}\left(U_{QFT}\otimes I\right)\left(\frac{1}{\sqrt{Q}}\sum_{j=0}^{Q-1}\left|j\right\rangle\left|f(j)\right\rangle\right) &= \frac{1}{\sqrt{Q}}\sum_{j=0}^{Q-1}\frac{1}{\sqrt{Q}}\sum_{k=0}^{Q-1} e^{(2\pi i k j)/Q}\left|k\right\rangle\left|f(j)\right\rangle \\ &= \frac{1}{Q}\sum_{j=0}^{Q-1}\sum_{k=0}^{Q-1} e^{(2\pi i k j)/Q}\left|k\right\rangle\left|f(j)\right\rangle\end{aligned} \tag{2.118}$$

Step 4: Measure the state.

At this stage, the probability of observing the state $\left|k\right\rangle\left|f(j)\right\rangle$ is given by

$$P(k, f(j)) = \left|\frac{1}{Q}\sum_{f(x)=f(j)} e^{(2\pi i k x)/Q}\right|^2 \tag{2.119}$$

where $0 \le x \le Q-1$. Because of the periodicity of *f* we have $x = j + br$, where *r* is the period of *f* and *b* is an integer. If we insert this into the equation (2.119), we get

$$P(k, f(j)) = \left| \frac{1}{Q} \sum_{b=0}^{[(Q-1-j)/r]} e^{(2\pi i k(j+br))/Q} \right|^2 . \tag{2.120}$$

Since we are interested in the relative magnitudes of the probabilities for observing different states we can factor out the term $e^{(2\pi ikj)/Q}$. Then we get

$$P(k, f(j)) \approx \left| \frac{1}{Q} \sum_{b=0}^{[(Q-1-j)/r]} e^{(2\pi ib(r,k))/Q} \right|^2 \tag{2.121}$$

where we define (r,k) so that it satisfies $-\frac{Q}{2} \le (r,k) \le \frac{Q}{2}$ and $(r,k) = rk (\mathrm{mod}\, Q-1)$. The probability distribution in (2.121) can be verified to have peaks where (r,k) is small with respect to *r* like in the case when *r* times *k* is a multiple of *Q*. Hence for the values of k with the highest probability of being observed, we have

$$rk = nQ \tag{2.122}$$

where *n* is an integer. Since we know the value of *Q* and since we can locate the peaks of the distribution by repeating the above steps, we can make use of (2.122) to find the period *r* by applying the continued fractions algorithm [32].

Since period finding can be solved in a polynomial number of steps, we have an efficient way of finding the order of an integer in any base. This means we can efficiently solve the factorization problem on quantum computers.

2.10. There is More to Quantum Computation

A variety of ideas from the relatively simple ones such as generation of truly random numbers to more complex ones such as quantum copying, can be addressed in relation to quantum computers. Some of these were discussed in the previous sections and quantum

simulation will be examined in detail in a later chapter. Of course, there is much more to quantum computation than discussed here. Let us now devote a few words to some of those interesting ideas which can be viewed to be in the field of quantum computation and quantum information theory.

Quantum communication and quantum cryptography seem to be the most interesting potential applications for quantum computation. In both fields, remarkable results have been demonstrated. In the communication domain, it is shown that it takes practically no time to transfer the quantum state information to any distance with use of the method known as quantum teleportation depending on the entanglement and no cloning principles. There is more: a technique known as dense coding can transmit two classical bits of information with use of only a single qubit. In the cryptography domain, things are not so simple. The best classical security measures depending on public key cryptography are only polynomially hard to break, with a quantum computer which enjoys the advantages of quantum factorization and discrete logarithms. On the other hand, a technique known as quantum key distribution is shown to provide provably secure and efficient distribution of information. These ideas are the source of most of the interest in quantum computation, and it is this interest which helps the development of the field.

There are less 'popular' ideas which are of extreme value if quantum computers are ever to be realized. Several physical models including ion traps, nuclear magnetic resonance, harmonic oscillator, nonlinear optics and cavity electrodynamics, are studied in detail as potential realizations of quantum computers or quantum information processing devices. The main difficulty is that superpositional states can hardly survive when they are interacting with the environment. (This phenomenon is known as decoherence.) As of today, only small scale systems could be constructed for experimental issues and hence the physical realization problem is still a challenging one. Another active field of research is quantum error correction which aims to protect quantum states from the effects of decoherence and noise. This is a crucial task for closing the gap between the theoretical and experimental development of quantum computation and information theory.

There are lots of other interesting application areas such as distributed quantum computation and quantum game theory. [33] presents a list of physically implementable quantum functionalities, from a quantum gate to a quantum robot. It can be expected that the list of such ideas may grow faster and faster as our intuition about the nature of quantum systems gets stronger.

For most of the topics mentioned above [4] is a valuable reference for readers of various levels. [6, 9, 34] are also informative in many aspects.

3. A SHORT SURVEY ON QUANTUM SIMULATION

Simulation is basically the task of calculating and describing a future state of a system whose initial state is given as an input. This task is generally achieved by an analysis of a set of differential equations which governs the time evolution of the state variables of the system. The simulation of a quantum system by classical apparatus is theoretically possible but highly inefficient to be useful. This is because the quantum mechanical state of a real system of a reasonable size requires an exponentially large number of variables just to be fully described. Analyzing the evolution of such a system also requires the multiplication of exponentially large matrices. This difficulty was, in fact, one of the first inspirations to make scientists think about the capabilities of quantum computers.

The simulation of quantum systems is the first classically hard problem for which quantum computers were expected to provide efficient means of solution. In a paper, he wrote in 1982, the famous physicist R.P. Feynman pointed out the practical impossibility of simulating quantum systems on classical computers, and suggested that quantum physics could be simulated with use of quantum computer elements [35]. He claimed that it was possible to build a universal quantum simulator whose state and evolution could be programmed to reproduce the behavior of any quantum system of interest.

This remained as a conjecture until 1996, when Lloyd [36], Wiesner [37], and Zalka [38] separately showed that conventional quantum computers could be programmed to efficiently perform universal quantum simulation of systems whose states evolve according to local interactions. Since then, quantum simulation is considered to be one of the three main applications of quantum computation, together with Shor's algorithm for factorization and Grover's algorithm for unstructured search. Of the three, quantum simulation seems to be the most promising one, since it forms a basis on which a variety of useful ideas can be implemented to provide meaningful performance gains against classical computers.

Before going into a detailed analysis, let us build a notion of simulation. There are two important decisions to be made for a successful simulation. The first one is related to the representation of the simulated system. One must decide how to map the state space of the simulated system to the state space of the simulator. The second decision is about the evolution operator to be used on the simulator. It should be decided which operator mimics the evolution of the simulated system best. These two decisions are strongly related to each other, and the correlation between them is a factor in determining the success of a simulation. Let us consider Figure 3.1, which depicts the correspondence between the simulator and the simulated system.

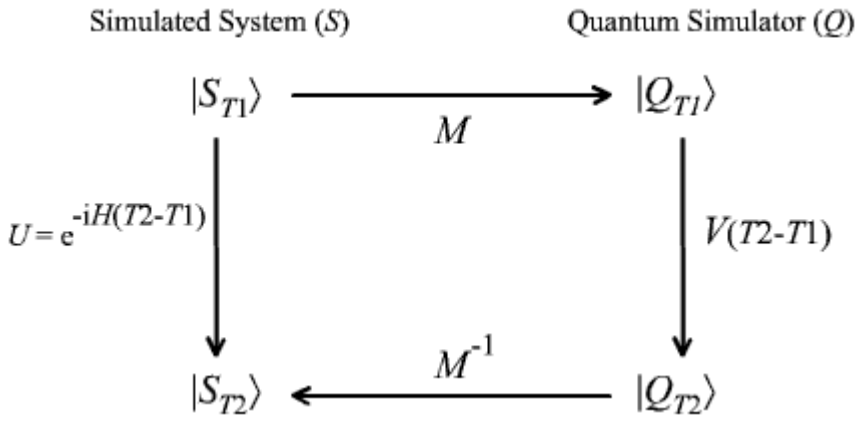

Figure 3.1. The correspondence between the simulator and the simulated system

Figure 3.1 suggests a scheme in which the states of the simulated system and the simulator are related by a mapping M. Therefore, the initial state $|S_{T1}\rangle$ of the simulated system is represented by $|Q_{T1}\rangle = M(|S_{T1}\rangle)$ on the simulator. The unitary mapping $V(T2-T1)$ mimics the evolution of the simulated system from time point $T1$ to time point $T2$, where this evolution is governed by the Hamiltonian H. The state on the simulator becomes $|Q_{T2}\rangle = V(T2-T1)|Q_{T1}\rangle$. Then the state of the simulated system at time $T2$ is guessed to be $M^{-1}(|Q_{T2}\rangle)$. The accuracy of this guess depends on how well M maps the states of the two systems and how closely $V(T2-T1)$ mimics the evolution of the simulated system

Note that, we generally use the phrase "simulating a Hamiltonian *H*" instead of "simulating the system governed by the Hamiltonian *H*". The phrase "simulating the evolution $e^{-iH\Delta t}$" is also equivalent with the additional remark on the duration of simulation Δt. In general, a system, which evolves according to the Hamiltonian H for a duration Δt, undergoes a transformation described by $e^{-iH\Delta t}$. If the state of the system before this transformation is defined by $|\psi(0)\rangle$ then at time $t = 0 + \Delta t$, the state of the system is described by $|\psi(t)\rangle = e^{-iH\Delta t}|\psi(0)\rangle$.

We present a sketch of Lloyd's analysis of quantum simulation in Section 3.1. Then in Section 3.2, we will examine what types of Hamiltonians can be simulated and will introduce several useful tools for quantum simulation. Section 3.3 briefly discusses the power of quantum simulation.

3.1. Universal Quantum Simulators

In [36], Lloyd defines a transformation $U = e^{iAt}$, where A lies in the algebra $\mathcal{A}$, generated by commutation from the set of Hamiltonians which correspond to time evolution of a system in various experimental setups, $\{\tilde{H}_1, \tilde{H}_2, .., \tilde{H}_\ell\}$. The complexity of simulating such a transformation can be measured by the number of elementary operations required to build this transformation. The number of operations required to generate an arbitrary $m \times m$ transformation, *U*, is of the order of m^2, which is number of parameters required to specify *U*.

To simulate such a transformation of *N* quantum variables on a classical computer requires the multiplication of a 2^N dimensional state vector by a $2^N \times 2^N$ matrix and computational time and memory space of the order 2^{2N}. A quantum computer can do the same within a memory space of *N* qubits, however the number of operations required to build an arbitrary U remains exponentially large even for a quantum computer. If a quantum system which evolves according to local interactions is chosen instead of an arbitrary quantum

system, then as Lloyd has shown, the simulation of the system on a quantum computer is much more efficient than the same on a classical computer.

Consider a quantum system of *N* variables, with Hamiltonian $\sum_{i=1}^{\ell} H_i$, where each H_i acts on a space of dimension m_i, encompassing at most k of the variables. Any Hamiltonian system with local interactions can be written in this form. The quantum simulation of such a system depends on the identity $e^{iHt} = \lim_{n\to\infty}\left(e^{iH_1 t/n}\cdots e^{iH_\ell t/n}\right)^n$, hence it works by evolving the system forward locally, over small, discrete time slices in loops.

The desired accuracy of simulation can be achieved by regulating the time slicing according to the equation

$$e^{iHt} = \left(e^{iH_1 t/n}\cdots e^{iH_\ell t/n}\right)^n + \sum_{i>j}\left[H_i, H_j\right]t^2/2n + \sum_{k=3}^{\infty} E(k), \tag{3.1}$$

where the higher order error terms $E(k)$ are bounded by $\|E(k)\|_{SUP} \le n\|Ht/n\|_{SUP}^k/k!$. ($[A,B]$ is the commutator of operators *A* and *B* given by $[A,B] = AB - BA$ and $\|A\|_{SUP}$ is the supremum or the maximum expectation value of the operator *A* over the states of interest.) The total error is less than $\left\|n(e^{iHt/n} - 1 - iHt/n)\right\|_{SUP}$, which can be made as small as desired by taking *n* sufficiently large. (3.1) suggests that the minimum number of steps n required to simulate the system to accuracy ε over time t is proportional to t^2/ε. Therefore Lloyd states that, for any $\varepsilon > 0$, one can pick a sufficiently large *n* to ensure that the simulator always tracks the correct time evolution e^{iHt} to within ε.

The quantum computational complexity of performing the simulation with a fixed accuracy can be estimated. Since each H_j acts on a local Hilbert space of only m_j dimensions, $e^{iH_j t/n}$ can be simulated with use of m_j^2 operations. The number of operations

needed to simulate the time evolution e^{iHt} to an accuracy ε, is approximately $n\sum_{i=1}^{\ell} m_i^2$. If m is taken to be the largest one of the m_j's then, one can insert $n\ell m_j^2$ as an upper bound on the number of operations needed to achieve the simulation within the desired accuracy.

The above simulation method is efficient (in the manner that it can be achieved with use of polynomially large computational resources) as long as $\ell = \ell(N)$ is a polynomial function of N. The total simulation time t is divided to n equal slices where *n* is proportional to t^2/ε. Then each coherent operation $e^{iH_j t/n}$, requires a duration proportional to t/n, that is, to $1/t$. It turns out that the total duration needed to simulate a system for time *t*, is just proportional to *t*. The level of desired accuracy affects only the number and (hence) the length of the slices into which the overall time is divided.

To sum up, the number of qubits to be used in the simulation of a quantum system is proportional to the size of that system and this simulation takes a time proportional to the duration over which the simulated system evolves. This concludes Lloyd's proof of Feynman's conjecture that quantum computers are able to simulate other quantum systems more efficiently than classical computers.

3.2. Constraints on the Simulation of Hamiltonian Dynamics

We have seen that not all Hamiltonians can be simulated efficiently even with the use of quantum computers. (This is in fact another way of saying that there exist unitary operations which quantum computers cannot efficiently approximate. Recall that there is a one to one relation between Hermitian and unitary operators defined by $U = e^{iH}$.) We can say that a Hamiltonian can be simulated if it defines the dynamics of a physical system that can be realized. However, this remark is of little use for practical computations. Hence, there is the need for expressing the same idea in a way compatible with the quantum circuit model of computation. Such an analysis can be found in [39], where a set of techniques for quantum simulation is listed. Here, we briefly introduce some of these tools to be used later .

3.2.1. Simulation of Local Hamiltonians

If a Hamiltonian H acts on a constant number of qubits, then it can be efficiently simulated. This is just another way of stating the fact that a unitary evolution on a constant number of qubits can be approximated with use of constant number of one and two qubit gates.

3.2.2. Rescaling of Hamiltonians

If a Hamiltonian H can be efficiently simulated, then cH can also be simulated efficiently as long as c is only polynomially big in terms of the size of the system that H acts son.

3.2.3. Unitary Conjugation

If a Hamiltonian H can be efficiently simulated and the unitary transformation U can be implemented efficiently, then it is possible to simulate the Hamiltonian $UHU^\dagger$ in an efficient manner. This follows from the simple identity

$$Ue^{-iHt}U^\dagger = e^{-iUHU^\dagger t}. \tag{3.2}$$

With use of unitary conjugation, it is possible to rotate the basis on which a Hamiltonian is applied.

3.2.4. Commutation

If two Hamiltonians H_1 and H_2 can be simulated efficiently, then the Hamiltonian $i[H_1, H_2]$ can also be efficiently simulated. ($[A, B]$ denotes the commutation of two operators defined as $[A,B] = AB - BA$. If $[A,B] = 0$, then A and B are said to be commuting operators.) This follows from the identity

$$e^{i[H_1,H_2]t} = \lim_{r\to\infty}\left(e^{-iH_1 t/\sqrt{r}} e^{-iH_2 t/\sqrt{r}} e^{iH_1 t/\sqrt{r}} e^{iH_2 t/\sqrt{r}}\right)^r . \tag{3.3}$$

3.2.5. Addition of Hamiltonians

If two Hamiltonians H_1 and H_2 can be simulated efficiently, then the Hamiltonian $H_1 + H_2$ can also be simulated efficiently. This is a result of the Trotter formula, a proof for which can be found in [4].

$$\lim_{r\to\infty}\left(e^{iH_1 t/r} e^{iH_2 t/r}\right)^r = e^{i(H_1+H_2)t} . \tag{3.4}$$

The following identities are also useful when approximating the sum of two Hamiltonians. Of these, (3.7) is known as Baker-Campbell-Hausdorf formula.

$$e^{i(H_1+H_2)t} = e^{iH_1 t} e^{iH_2 t} + O(t^2) . \tag{3.5}$$

$$e^{i(H_1+H_2)t} = e^{iH_1 t/2} e^{iH_2 t} e^{iH_1 t/2} + O(t^3) \tag{3.6}$$

$$e^{i(H_1+H_2)t} = e^{iH_1 t} e^{iH_2 t} e^{-\frac{1}{2}[H_1,H_2]t^2} + O(t^3) \tag{3.7}$$

3.2.6. Linear Combination of Hamiltonians

As a generalization of the addition rule, we can state that if a Hamiltonian H is a sum of polynomially many Hamiltonians $H_1,..,H_q$, each of which can be simulated efficiently, then H can also be simulated efficiently. This depends on the same argument as Lloyd's analysis. Let us see how this argument is built.

The Taylor expansion of the term e^x is given as $e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!}$. Then we can write

$$\begin{aligned} e^{-iH\Delta t} &= I + (-iH\Delta t) + \frac{(-iH\Delta t)^2}{2!} + ... \\ &= I - iH\Delta t - \frac{(H\Delta t)^2}{2} + ... \end{aligned} \tag{3.8}$$

If $H = H_1 + H_2 + ... + H_q$, assuming Δt is small we can write

$$\begin{aligned} e^{-i\left(H_1+H_2+...+H_q\right)\Delta t} &= I - i\left(H_1 + H_2 + ... + H_q\right)\Delta t - \frac{\left(H_1 + H_2 + ... + H_q\right)^2 (\Delta t)^2}{2} + O\left((\Delta t)^3\right) \\ &= I - i\left(H_1 + H_2 + ... + H_q\right)\Delta t - \left(H_1^2 + H_2^2 + ... + H_q^2\right)\frac{(\Delta t)^2}{2} \\ &\quad - \left(\sum_{1\le i<j\le q} H_i H_j\right)\frac{(\Delta t)^2}{2} - \left(\sum_{1\le i<j\le q} H_j H_i\right)\frac{(\Delta t)^2}{2} + O\left((\Delta t)^3\right) \end{aligned} \tag{3.9}$$

The product $e^{-iH_1\Delta t}e^{-iH_2\Delta t}\cdots e^{-iH_q\Delta t}$ can be written as

$$\begin{aligned} & e^{-iH_1\Delta t}e^{-iH_2\Delta t}\cdots e^{-iH_q\Delta t} \\ &= \left(I - iH_1\Delta t - \frac{(H_1\Delta t)^2}{2} + ...\right)\left(I - iH_2\Delta t - \frac{(H_2\Delta t)^2}{2} + ...\right)...\left(I - iH_q\Delta t - \frac{(H_q\Delta t)^2}{2} + ...\right) \\ &= I - i\left(H_1 + H_2 + ... + H_q\right)\Delta t - \left(\sum_{1\le i<j\le q} H_i H_j\right)(\Delta t)^2 - \left(H_1^2 + H_2^2 + ... + H_q^2\right)\frac{(\Delta t)^2}{2} + O\left((\Delta t)^3\right) \end{aligned} \tag{3.10}$$

Then the difference $e^{-i\left(H_1+H_2+...+H_q\right)\Delta t} - \left(e^{-iH_1\Delta t}e^{-iH_2\Delta t}\cdots e^{-iH_q\Delta t}\right)$ can be calculated as

$$\begin{aligned} e^{-i\left(H_1+H_2+...+H_q\right)\Delta t} - \left(e^{-iH_1\Delta t}e^{-iH_2\Delta t}\cdots e^{-iH_q\Delta t}\right) &= \left(\sum_{1\le i<j\le q} H_i H_j\right)\frac{(\Delta t)^2}{2} - \left(\sum_{1\le i<j\le q} H_j H_i\right)\frac{(\Delta t)^2}{2} + O\left((\Delta t)^3\right) \\ &= \left(\sum_{1\le i<j\le q} H_i H_j - H_j H_i\right)\frac{(\Delta t)^2}{2} + O\left((\Delta t)^3\right) \\ &= \left(\sum_{1\le i<j\le q} \left[H_i, H_j\right]\right)\frac{(\Delta t)^2}{2} + O\left((\Delta t)^3\right) \end{aligned} \tag{3.11}$$

Taking the norms and ignoring $O\left((\Delta t)^3\right)$ gives

$$\begin{aligned}\left\| e^{-i\left(H_1+H_2+\ldots+H_q\right)\Delta t} - \left(e^{-iH_1\Delta t}e^{-iH_2\Delta t}\cdots e^{-iH_q\Delta t}\right)\right\| &= \left\| \left(\sum_{1\le i<j\le q}\left[H_i,H_j\right]\right)\frac{(\Delta t)^2}{2}\right\| \\ &\le \left(\sum_{1\le i<j\le q}\left\|\left[H_j,H_i\right]\right\|\right)\frac{(\Delta t)^2}{2}\end{aligned} \tag{3.12}$$

If we let $\ell = \max_{i,j}\left(\left\|\left[H_i,H_j\right]\right\|\right)$ then we can write

$$\left\| e^{-i\left(H_1+H_2+\ldots+H_q\right)\Delta t} - \left(e^{-iH_1\Delta t}e^{-iH_2\Delta t}\cdots e^{-iH_q\Delta t}\right)\right\| = O\left(q^2\ell(\Delta t)^2\right) \tag{3.13}$$

since the summation on the right hand side of (3.12) contains $O\left(q^2\ell(\Delta t)^2\right)$ terms. It is also useful to note that the product of the terms $e^{-iH_1\Delta t}, e^{-iH_2\Delta t}, \cdots, e^{-iH_q\Delta t}$ could be arranged in any order and still satisfy (3.13).

Now let a denote the product $\left(e^{-iH_1\Delta t}e^{-iH_2\Delta t}\cdots e^{-iH_q\Delta t}\right)$ and b denote the error term $O\left(q^2\ell(\Delta t)^2\right)$. So we can write $e^{-i\left(H_1+H_2+\ldots+H_q\right)\Delta t} = a+b$. Then

$$\begin{aligned}\left(e^{-i\left(H_1+H_2+\ldots+H_q\right)\Delta t}\right)^r &= (a+b)^r \\ &= \sum_{k=0}^{r}\binom{r}{k}a^{r-k}b^k \\ &= a^r + ra^{r-1}b + \ldots \\ &= a^r + O(rb) \\ &= \left(e^{-iH_1\Delta t}e^{-iH_2\Delta t}\cdots e^{-iH_q\Delta t}\right)^r + O\left(rq^2\ell(\Delta t)^2\right)\end{aligned} \tag{3.14}$$

Now, if we let $t = r\Delta t$ then we get the rule known as Lie product formula.

$$\left\| e^{-i\left(H_1+H_2+\ldots+H_q\right)t} - \left(e^{-iH_1 t/r} e^{-iH_2 t/r} \cdots e^{-iH_q t/r} \right)^r \right\| = O\left(\frac{q^2 \ell t^2}{r} \right) \tag{3.15}$$

In the limit case where r goes to infinity, we have

$$\lim_{r\to\infty} \left(e^{iH_1 t/r} \cdots e^{iH_q t/r} \right)^r = e^{i\left(H_1+..+H_q\right)t} \tag{3.16}$$

It is also possible to combine (3.16) with the rule about rescaling to produce a more general result known as linear combination of Hamiltonians: If a Hamiltonian H can be represented as $\sum_{i=1}^{q} c_i H_i$, where for $1 \le i \le q$, c_i is a real number polynomially large, and H_i is a Hamiltonian which can be simulated efficiently, then H can also be simulated efficiently.

$$\lim_{r\to\infty} \left(e^{ic_1 H_1 t/r} \cdots e^{ic_q H_q t/r} \right)^r = e^{i\left(c_1 H_1+..+c_q H_q\right)t} \tag{3.17}$$

3.2.7. Tensor Products

If each of the Hamiltonians $H_1,..,H_q$ acts on a constant number of qubits and the product of their eigenvalues is efficiently computable, then the Hamiltonian $H_1 \otimes .. \otimes H_q$ can be efficiently simulated. This is done with a technique to be examined in Section 3.3.2.

3.3. Quantum Simulation in Algorithmic Level

In this section, we will analyze quantum simulation in algorithmic level. Section 3.3.1 will introduce an algorithm for quantum simulation of systems whose states evolve according to local interactions. Then in Section 3.3.2, the simulation of Hamiltonians which act non-trivially on all or nearly all parts of a large system will be considered. Our main reference for both topics is [4].

3.3.1. An Algorithm for the Simulation of Local Interactions

In Section 3.1, we had briefly mentioned a process that Lloyd had suggested for the simulation of systems which evolve according to local interactions. The following simulation algorithm, which can be found in [4], depends on similar ideas.

The algorithm runs on the following inputs:

1. A Hamiltonian $H = \sum_k H_k$, which acts on a system whose state lies in an *N*-dimensional Hilbert space and where each H_k acts on a small subsystem of size independent of *N*.

2. An initial state $|\psi_0\rangle$, of the system at time $t = 0$.

3. A positive accuracy ε.

4. A time t_f at which the evolved state is desired.

The algorithm to simulate *H* on these inputs is defined as follows:

Choose a representation such that the state $|\tilde{\psi}\rangle$ of *n* qubits (where *n* is polynomially big in terms of the logarithm of *N*, $n = poly(\log N)$) approximates the system, and the operators $e^{-iH_k \Delta t}$ have efficient quantum circuit approximations. Select an approximation method (consider for example (3.4-7) and (3.13)) and time slicing Δt, such that the expected error is acceptable. Construct the corresponding quantum circuit for the iterative step $U_{\Delta t}$ and do:

1. $|\tilde{\psi}_0\rangle \leftarrow |\psi_0\rangle$; $j = 0$ .....initialize state
2. $\rightarrow |\tilde{\psi}_{j+1}\rangle = U_{\Delta t}|\tilde{\psi}_j\rangle$ .....iterative update
3. $\rightarrow j = j + 1$; goto 2 until $j\,\Delta t \geq t_f$ .....loop structure
4. $\rightarrow |\tilde{\psi}(t_f)\rangle = |\tilde{\psi}_j\rangle$ .....final result

The procedure described above needs $O(poly(1/\varepsilon))$ operations and it outputs a state $|\tilde{\psi}(t_f)\rangle$ such that $\left|\langle\tilde{\psi}(t_f)|e^{-iHt_f}|\psi_0\rangle\right|^2 \geq 1-\delta$.

As an example, consider the simulation of a Hamiltonian H which can be written as a sum $H_1+H_2+H_3$ of three Hamiltonians, where H_1, H_2 and H_3 can be efficiently simulated so that it is possible to build polynomial size circuits for approximating e^{-iH_1t}, e^{-iH_2t} and e^{-iH_3t} for arbitrary t. Suppose the total simulation time is τ and suppose we choose (3.13) as the approximation method. First, we define the number of slices *r* that satisfies the accuracy constraints on our simulation by considering the structure of the error term in (3.15). Then we get the length for each slice: $\Delta t = \tau/r$. Next we need to construct the circuits for $e^{-iH_1\Delta t}$, $e^{-iH_2\Delta t}$ and $e^{-iH_3\Delta t}$, which are combined together to construct the quantum circuit for the iterative step $U_{\Delta t} = e^{-iH_1\Delta t}e^{-iH_2\Delta t}e^{-iH_3\Delta t}$. What remains is to apply $U_{\Delta t}$, *r* times on the initial state of the system. The circuit in Figure 3.2 demonstrates this process.

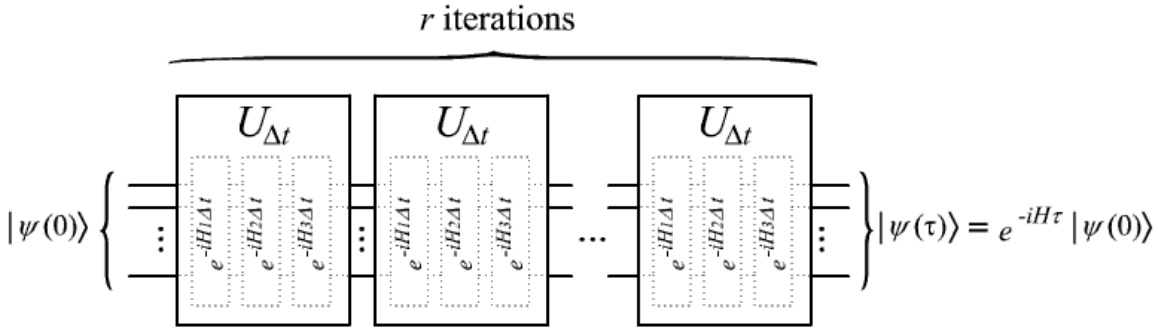

Figure 3.2. The circuit for simulating $H = H_1 + H_2 + H_3$ for duration $\tau = r\Delta t$

3.3.2. Simulation of Hamiltonians with Non-local Terms

The algorithm introduced in Section 3.3.1 is concentrated on simulating Hamiltonians which are sums of local interactions, however, this is not a fundamental requirement for a Hamiltonian to be efficiently simulated. It is also possible to efficiently simulate Hamiltonians which act non-trivially on all or nearly all parts of a large system.

Consider the Hamiltonian $H = \bigotimes_{1}^{n} Z$ which acts on a system of *n* qubits where *Z* is the Pauli *Z* operator given by

$$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}. \tag{3.18}$$

The interaction defined by $e^{-iH\Delta t}$, (where $H = \bigotimes_{1}^{n} Z$) involves the whole of the system, but it can still be efficiently simulated because it does so in a classical manner: it applies a phase shift of either $e^{-i\Delta t}$ or $e^{i\Delta t}$ to the system. It is the total parity of the *n* input qubits in the computational basis which determines the exact shift to be applied. If the total parity is even then the phase shift applied to the system is $e^{-i\Delta t}$, and if the total parity is odd then the phase shift is $e^{i\Delta t}$. This can be trivially seen in the case where *n* is taken to be 3. Let us consider $H = \bigotimes_{1}^{3} Z$ given in (3.19).

$$Z \otimes Z \otimes Z = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \end{bmatrix}. \tag{3.19}$$

Note that the diagonal entries d_i in the above matrix are 1 if the total parity of the bits which represent *i* in binary format is even and they are -1 if the total parity is odd. d_0 for example is 1 since 0 is represented by "000" in binary format, which has even total parity while d_4 is -1 since 4 is represented by "100" in binary format, which has odd total parity of bits. Also note that the same property is reflected in the operator $e^{-iH\Delta t}$ given in (3.20).

$$e^{-iH\Delta t} = e^{-i(Z\otimes Z\otimes Z)\Delta t} = \begin{bmatrix} e^{-i\Delta t} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & e^{i\Delta t} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & e^{i\Delta t} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & e^{-i\Delta t} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & e^{i\Delta t} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & e^{-i\Delta t} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & e^{-i\Delta t} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & e^{i\Delta t} \end{bmatrix}. \qquad (3.20)$$

So if the three input qubits have even total parity in the computational basis (like in the cases for "000", "011", "101", "110") the phase shift to be applied is $e^{-i\Delta t}$, that is $e^{-iH\Delta t}|\psi\rangle = e^{-i\Delta t}|\psi\rangle$, and if the total parity is odd (like in the cases for "001", "010", "100", "111") the phase shift to be applied is $e^{-i\Delta t}$, that is $e^{-iH\Delta t}|\psi\rangle = e^{i\Delta t}|\psi\rangle$. The reader should also be convinced that this is not specific to the case where $n = 3$, but it is valid for any value of n.

So in order to simulate the evolution $e^{-iH\Delta t}$, there is the need for a circuit which first calculates the total parity of the input qubits in the computational basis and then applies the appropriate phase shift as described above. The circuit in Figure 3.3 simulates the Hamiltonian $H = \bigotimes_1^3 Z$ for the duration Δt on three qubits initially prepared in the state $|\psi(0)\rangle$ with the use of an ancilla qubit initially prepared in the state $|0\rangle$. The circuit starts by classically computing the total parity of the three input qubits in the computational basis with the help of the CNOT gates and it stores the result in the ancilla qubit. Once the parity is stored on a single qubit, the simplest way of applying the necessary phase shift to the system is simulating the Hamiltonian $H = \bigotimes_1^1 Z$ on this qubit. (This can be conceived as reducing the problem of simulating $Z\otimes Z\otimes Z$ on three qubits to the problem of simulating Z on a single qubit.) So the circuit does this; it simulates Z on the ancilla qubit for the duration Δt and hence applies the necessary phase shift to the ancilla bit and thus to the system. The next step

is uncomputing the ancilla bit and this is all the circuit does. This strategy is clearly extendible to any value of *n*.

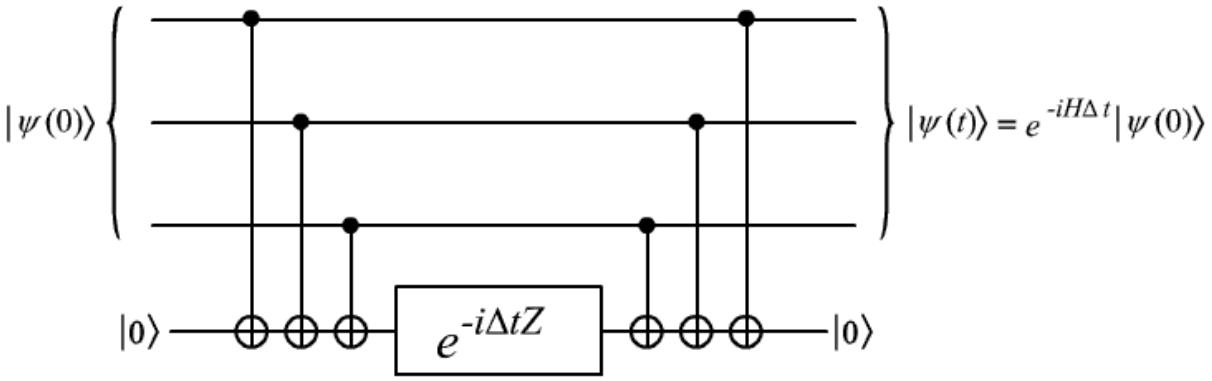

Figure 3.3. The circuit for simulating $H = Z \otimes Z \otimes Z$ for duration Δt

By extending the above procedure, it may also be possible to simulate more complicated extended Hamiltonians. In particular, the procedure can be applied to any Hamiltonian of the form $H = \bigotimes_{k=1}^{n} \sigma_k$, where σ_k is a Pauli operator or the identity operator acting on the *k*'th qubit. By disregarding the qubits upon which the identity is applied and transforming *X* and *Y* terms to *Z* operations with the help of single qubit gates, the Hamiltonian can be put into the form of $H = Z \otimes Z \otimes .. \otimes Z$, which can be simulated as described above. The next example will clarify the idea.

Consider the Hamiltonian $H = X \otimes Y \otimes Z$ which acts on a system of 3 qubits where *X, Y, Z* are the Pauli operators given by

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, \quad Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}. \tag{3.21}$$

In order to simulate *H* with use of the technique introduced in Section 3.3.2, there is the need for expressing the *X* and *Y* operators in terms of *Z*. This can be conceived as implementing these operators in the *Z* basis. Simple linear algebra is of help here. The spectral

decomposition theorem states that every Hermitian matrix A can be decomposed into the form $A = U\Lambda U^{-1}$, where U is unitary and Λ is a diagonal matrix of the same size as A. Let us consider the spectral decompositions of X and Y.

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = U_x \Lambda_x U_x^{-1} = \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ 1/\sqrt{2} & -1/\sqrt{2} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ 1/\sqrt{2} & -1/\sqrt{2} \end{bmatrix} \tag{3.22}$$

$$Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} = U_y \Lambda_y U_y^{-1} = \begin{bmatrix} -i/\sqrt{2} & i/\sqrt{2} \\ 1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} i/\sqrt{2} & 1/\sqrt{2} \\ -i/\sqrt{2} & 1/\sqrt{2} \end{bmatrix} \tag{3.23}$$

Both Λ_x and Λ_y are equivalent to Z. The operators U_x and U_y and their inverses are all 2 by 2 and unitary matrices, and hence can be implemented as single bit quantum gates. Therefore we can simulate $H = X \otimes Y \otimes Z$ (for duration Δt) by first transforming into Z basis, then simulating $H = Z \otimes Z \otimes Z$ (for duration Δt), and then returning to the original basis. This is actually a result of the following identity:

$$\begin{aligned} H\Delta t &= (X \otimes Y \otimes Z)\Delta t \\ &= \left((U_x Z U_x^{-1}) \otimes (U_y Z U_y^{-1}) \otimes (IZI)\right)\Delta t \\ &= \left((U_x \otimes U_y \otimes I)(Z \otimes Z \otimes Z)(U_x^{-1} \otimes U_y^{-1} \otimes I)\right)\Delta t \\ &= (U_x \otimes U_y \otimes I)((Z \otimes Z \otimes Z)\Delta t)(U_x^{-1} \otimes U_y^{-1} \otimes I) \end{aligned} \tag{3.24}$$

Therefore, we should first apply $U_x^{-1} \otimes U_y^{-1} \otimes I$ to transform into Z basis, then simulate $H = Z \otimes Z \otimes Z$ for duration Δt as described in the previous example and then return to the original basis by applying $U_x \otimes U_y \otimes I$. The circuit in Figure 3.4 implements this idea to simulate $H = X \otimes Y \otimes Z$ for a duration Δt.

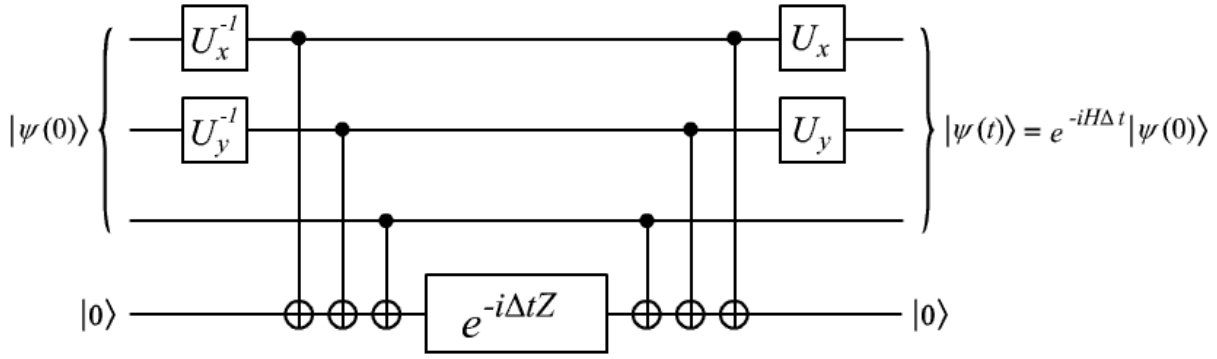

Figure 3.4. The circuit for simulating $H = X \otimes Y \otimes Z$ for duration Δt

A wide class of Hamiltonians with non-local terms can be simulated with this method. In particular, it is possible to efficiently simulate a Hamiltonian of the form $H = \sum_{k=1}^{L} H_k$, as long as L is polynomial in terms of the total number of particles in the system, and each individual H_k can efficiently be simulated by a quantum circuit.

3.4. Quantum Simulation as a Model of Quantum Computation

We have briefly seen that simulation of quantum systems is one of the tasks for which quantum computers are superior to classical computers. This result is important in itself but it raises further important questions about the potential applications of the quantum simulation ideas. Let us put aside the implications of quantum simulation on the areas such as physics and chemistry and concentrate on what it means for computation.

A real quantum system can be efficiently simulated with the use of quantum simulation methods introduced in the previous sections if there is information about the Hamiltonian which governs it. This is rarely the case for real systems which are not of very small scales. On the other hand, an artificially designed system and its Hamiltonian can be of interest especially if they present a way of implementing some key algorithmic process. In other words, it is possible to design quantum algorithms in terms of continuous interpretation of the time evolution which is described by the Schrödinger equation and a specified Hamiltonian *H*.

Then, we can run these algorithms by simply simulating H on a quantum computer. This can be better understood if we recall that in the quantum circuit model, an algorithm is represented by a unitary operator U which is nothing but the evolution operator defined by some Hamiltonian H and some duration Δt as $U = e^{-iH\Delta t}$.

Therefore, any problem computable on a quantum circuit can also be computed by the simulation of some Hamiltonian H on the appropriately prepared quantum states for some duration Δt. And since the quantum circuit model is universal for computation, we can say that every computable problem is associated to some Hamiltonian as a practical means for its solution. There are important examples where this technique is of extreme use even though it is not always simple to represent an algorithmic idea in terms of continuous quantum dynamics.

In Section 3.2, we had stated that only the Hamiltonians of physically realizable systems can be simulated efficiently on a quantum computer. Now we have also stated that any computable problem can be solved by simulation of some Hamiltonian. This relation between physically realizable systems and computable problems is one of the main insights of quantum computation.

As a last remark, we should note that, so far we have considered simulations of only time independent Hamiltonians and stated the universality of these Hamiltonians for computation. Time dependent Hamiltonians can also serve as useful tools for computation. The techniques depending on the simulation of time dependent Hamiltonians are known as adiabatic quantum computation. These techniques have recently been the focus of increasing interest and several important results (including the equivalence to the quantum circuit model) have been obtained. A detailed discussion of the topic is out of the scope of the current work, however see [17, 40] for more about adiabatic quantum computation.

4. A SHORT SURVEY OF QUANTUM RANDOM WALKS

In classical computer science, random walks have been extensively studied in various contexts and for various tasks such as modeling of physical systems or graph coloring. Random walks are found useful for many tasks because by use of simple local transitions, they provide general means of exploring large numbers of combinatorial structures. It is natural to ask whether quantum versions of random walk algorithms can also provide similar advantages. This question has recently been the source of a considerable amount of research. Quantum random walks have been shown to have different characteristics than their classical counterparts, and several quantum algorithms that depend on quantum random walks have been devised.

In this chapter, we aim to provide an introductory description of the fundamental properties of quantum random walks. In doing this, we will mostly refer to definitions and results from [41 - 44]. In the following, we first briefly introduce classical random walks in Section 4.1. Then in Section 4.2, we will analyze how to implement the idea of a random walk idea on a quantum computer. Section 4.3 will conclude this discussion by briefly mentioning some of the important algorithmic results about quantum random walks.

4.1. Classical Random Walks

A classical random walk is most generally a model for random motion constrained by the structure of a graph. An imaginary walker can be thought to be placed on one of the vertices of the graph depending on a probability distribution which can be taken to describe the global state of the motion. The walk starts with an initial probability distribution on the vertices, which generally specifies a single starting node. The motion can be analyzed in discrete or continuous time. Most generally, at a time, the walker either stays where it is or moves to one of the neighbors of the vertex it is on, depending on some probabilistic transition operator which reflects the structure of the graph. The overall process can be thought of as the

evolution of the probability distribution which defines the state of the walk. This evolution converges to a stationary distribution after a while.

4.1.1. Discrete Time Classical Random Walk

In a discrete time random walk process, motion takes place in discrete time steps where the walker is expected to move to one of the adjacent vertices with equal probabilities for each. On an undirected graph $G = (V(G), E(G))$ with N vertices, this motion is defined by the $N \times N$ stochastic operator K, the general term for which is given by (4.1) where $d(a)$ denotes the degree of the vertex a.

$$k_{ab} = \begin{cases} \frac{1}{d(a)} & (a,b) \in E(G) \\ 0 & otherwise \end{cases} \tag{4.1}$$

The state after the n'th discrete time step is defined by the probability distribution vector $P(n)$ which is related to the initial state $P(0)$ through (4.2).

$$P(n) = K^n P(0) \tag{4.2}$$

4.1.2. Continuous Time Classical Random Walk

Following [44], we define a continuous time classical random walk on an undirected graph $G = (V(G), E(G))$ with N vertices and no self loops. The probability of jumping to any adjacent vertex in a time ε is defined to be $\gamma\varepsilon$ in the limit case where $\varepsilon \to 0$ and where γ is non-negative. The walk is defined by an $N \times N$ transition operator K, which is generally known as the infinitesimal generator matrix of the random walk. The off-diagonal entries $k_{a,b}$ of K are related to the probability of transition between the vertices a and b. Therefore, if there is an edge between a and b, then $k_{a,b} = \gamma$, and otherwise, $k_{a,b} = 0$. The diagonal entries $k_{a,a}$

are set to $-d(a)\gamma$ to satisfy normalization requirements, where $d(a)$ denotes the degree of the vertex a. Then the general term for the infinitesimal generator matrix is given in (4.3). (The reader may have noted that *K* is in fact equal to γ times the Laplacian matrix (*L*) of the graph *G*. $L = A - D$, where *A* is the adjacency matrix of the graph and *D* is a diagonal matrix with entries $d_{ii} = d(i)$)

$$k_{ab} = \begin{cases} \gamma & i(a,b) \in E(G) \\ -d(a)\gamma & a = b \\ 0 & otherwise \end{cases} \tag{4.3}$$

Let the *N*-dimensional vector $P(t)$ denote the probability distribution among vertices of the graph at time *t*. Then the *n*'th component, $p_n(t)$, of $P(t)$ shows the probability of the walker being at the *n*'th vertex of the graph at time *t*. The probability distributions $P(t)$ and $P(t+\varepsilon)$, where $\varepsilon \to 0$, are related by (4.4).

$$\lim_{\varepsilon \to 0} P(t+\varepsilon) = P(t) + K\varepsilon P(t) \tag{4.4}$$

Then the continuous evolution of the probability distribution can be said to be governed by (4.5).

$$\frac{d}{dt}P(t) = \lim_{\varepsilon \to 0}\left(\frac{P(t+\varepsilon) - P(t)}{\varepsilon}\right) = \frac{K\varepsilon P(t)}{\varepsilon} = KP(t) \tag{4.5}$$

Therefore the probability $p_a(t)$ of being at vertex *a* at time *t* evolves according to

$$\frac{d}{dt}p_a(t) = \sum_b k_{ab}p_b(t) = k_{aa}p_a(t) + \sum_{b \neq a} k_{ab}p_b(t)\,. \tag{4.6}$$

Since the diagonal entries k_{aa} of the generator matrix are set to $-d(a)\gamma$, which is equal to the negative of the sum of the other entries in the same column, the rows and columns of the matrix sum to zero. Then the total change in the probabilities for being at individual vertices can be shown to be zero as in (4.7). This is why an initially normalized distribution remains normalized. For all values of *t,* it holds that $\sum_a p_a(t) = 1$.

$$\begin{aligned} \sum_{a \in V(G)} \frac{d}{dt} p_a(t) &= \sum_{a \in V(G)} \left(k_{aa} p_a(t) + \sum_{b \neq a} k_{ab} p_b(t) \right) \\ &= \sum_{a \in V(G)} \left(k_{aa} p_a(t) + \sum_{b \neq a} k_{ab} p_a(t) \right). \\ &= 0 \end{aligned} \tag{4.7}$$

4.1.3. Performance Measures for Classical Random Walks

The random walk on *G,* after a while, converges to a stationary probability distribution $P(s)$, which is independent of the initial state. The speed of convergence for a random walk process is an important parameter that determines the efficiency of implementing it. It is possible to define several measures of the convergence of a random walk process. One of these measures, for example, is named *mixing time* and is defined as the smallest time duration required for the walk to ultimately reach to some ε neighborhood of the stationary state with respect to some measure of distance. The mixing time of a random walk is strongly related to the spectral properties of the generator matrix for it. Other measures of convergence include the *filling time* and *dispersion time*. See [41] for more details about these measures. Another useful performance measure is named *hitting time*. The hitting time h_{uv} of node *v* is the expected time it takes until the walk, starting from node *u,* hits *v* for the first time [43]. Different measures can be good for measuring the performance of random walks for different kinds of problems. Hitting time, for example, is critical for the problems like graph connectivity and k-SAT, while mixing time is an important measure for many tasks like sampling from a set of combinatorial structures.

4.2. Quantum Random Walks

Transforming the notion of a classical random walk to the notion of a quantum random walk is relatively simpler in the continuous case than it is in the discrete case. In the continuous case, it is of much help to note the similarity between the state of a random walk defined by a probability distribution vector and the state of a quantum system defined by a unit ket vector. It is further important to note the similarity in the differential equations governing the evolution of these systems. However, in the discrete case, there is the need for introducing some new ideas in addition to the same basic similarities. The idea will be better understood in the following sections, where we will analyze the quantum walk on simple graphs.

Let us consider the simpler case first and begin with the formulation of the continuous time quantum random walk in Section 4.2.1. Then in Section 4.2.2, we will analyze how to implement the discrete time quantum random walk.

4.2.1. Continuous Time Quantum Random Walks

A continuous time random walk on an undirected graph $G = (V(G), E(G))$ with N vertices and no self loops is classically implemented by the simple evolution of the state vector $|P(t)\rangle = \sum_{k=1}^{N} p_k(t)|k\rangle$ according to the equation $\frac{d}{dt}|P(t)\rangle = K|P(t)\rangle$ where K is the infinitesimal generator matrix introduced in Section 4.1.3, and $p_k(t)$ stands for the non-negative real probability value of being at node k at time t. Since the probabilities should sum up to one at any time the $p_k(t)$s should satisfy $1 = \sum_{k=1}^{N} p_k(t)$.

If we revise the above statement according to the principles of continuous quantum dynamics, we get the definition for a continuous time quantum random walk on the graph $G = (V(G), E(G))$. Let us do so. First we replace the probability distribution vector which lies in an n dimensional real space with the state of a quantum system which lies in an n

dimensional Hilbert space. We let the basis states $|1\rangle,|2\rangle,..,|N\rangle$ of this space to respectively represent the states of being at the nodes $v_1, v_2, .., v_N$ of the graph *G*. Therefore the general state vector is of the form $|\psi(t)\rangle = \sum_{k=1}^{N} \alpha_k(t)|k\rangle$, where $\alpha_k(t)$ represents the complex amplitude of the base state $|k\rangle$ at time *t*. This time, the normalization constraint is $1 = \sum_{k=1}^{N} |\alpha_k(t)|^2$.

Next, we need a Hamiltonian *H* to describe the evolution of the system. The infinitesimal generator matrix *K* of the continuous time classical random walk is a good candidate to serve as the Hamiltonian of our system. (Note that the generator matrix, $K = \gamma L$, is Hermitian.) However, this is not the only choice. In the classical case, the normalization constraint was $1 = \sum_{k=1}^{N} p_k(t)$, however, it is $1 = \sum_{k=1}^{N} |\alpha_k(t)|^2$ in the quantum case. This constraint is guaranteed to be satisfied under any unitary transformation, hence, we can choose any Hermitian matrix that reflects the structure and locality of the graph as the Hamiltonian. Simply using γ times the adjacency matrix of the graph as the Hamiltonian is therefore another choice. (The two different Hamiltonians we suggest here are not equivalent as long as the graph is not regular, that is, not all vertices have the same degree.)

$$H_{i,j} = \langle i|H|j\rangle = \begin{cases} \gamma & (v_i, v_j) \in E(G) \\ 0 & otherwise \end{cases} \tag{4.8}$$

If we let $\gamma = 1$ for simplicity, this becomes

$$H_{i,j} = \langle i|H|j\rangle = \begin{cases} 1 & (v_i, v_j) \in E(G) \\ 0 & otherwise \end{cases} \tag{4.9}$$

Finally, we let this system evolve according to the Schrödinger equation $i\hbar \frac{d}{dt}|\psi(t)\rangle = \mathrm{H}|\psi(t)\rangle$. Therefore running the continuous time random walk for duration Δt is

simply equivalent to simulating H for duration $\Delta t : |\psi(t+\Delta t)\rangle = e^{-iH\Delta t}|\psi(t)\rangle$. What remains is to implement this computation efficiently, and this can be done by simulation methods.

Let us now consider the continuous time random walk on a straight line graph (G) of eight nodes as depicted in Figure 4.1. Each of the edges in the graph have equal transition probabilities, hence the graph is said to be translationally invariant.

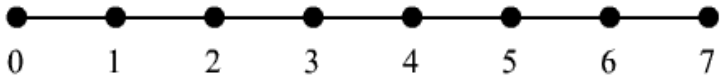

Figure 4.1. The straight line graph of eight nodes

We have two candidate operators to serve as the Hamiltonian. These are $H_1 = \gamma L$ and $H_2 = \gamma A$ where L denotes the Laplacian matrix and A denotes the adjacency matrix of the graph G. These operators are given in (4.10), where for simplicity γ is taken to be 1.

$$H_1 = \gamma L = \begin{bmatrix} -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -2 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -2 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix}, \quad H_2 = \gamma A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \tag{4.10}$$

Therefore, the continuous random walk on the straight line of eight nodes can be implemented on a quantum computer by simulating either the transformation $e^{-iH_1\Delta t}$, or the transformation $e^{-iH_2\Delta t}$. Both ways serve equally well even though they are not equivalent. (Note that G is not regular.)

4.2.2. Discrete Time Quantum Random Walks

Beginning with an example will be helpful in understanding why building a quantum model for discrete time random walks is not so straightforward. Then we will see how such a quantum model can be built on a state space other than the vertices of the graph. Last, we will see how this model can be generalized to make it useful for a larger set of graphs.

Let us consider the discrete time classical random walk on the straight line graph shown in Figure 4.1. Note that the vertices of the graph are labeled so that $n \to (n+1)$ is a transition to the right for $0 \le n \le 6$, $n \to (n-1)$ is a transition to the left for $1 \le n \le 7$, and there is no other choice. So if at a time step the walker is on a node *n* between 1 and 6, then in the next step it moves to left with probability 0.5 and to the right with probability 0.5. A straightforward quantum model for this step would be

$$|n\rangle \to \frac{1}{\sqrt{2}}\left(|n-1\rangle + |n+1\rangle\right). \tag{4.11}$$

The transition in (4.11) is not unitary however. (This can be verified by noting that orthogonal states like $|n\rangle$ and $|n+2\rangle$ are mapped to non-orthogonal states by this transition.) It can be shown that no discrete time quantum random walk on the vertices of a straight line (of any dimension) can be defined. This result can be extended to include many other graphs with the exception of some special cases.

There are several ways to implement discrete time quantum walks. All of these depend primarily on some extension of the state space. One such method for implementing a discrete time quantum walk on a straight line can be found in [45]. Let us introduce this method, which is built as an analogue of the real-life process where somebody flips a real coin to decide whether to go left or right. (Also note that, an infinite straight line graph is assumed in this process.)

We define the quantum walk on a state space spanned by the states of the form $|n,c\rangle$, where n is the usual register that represents the vertices of the graph and c is a bit we may name as a coin. A step of the quantum walk consists of two operations. First, we 'flip the coin' by applying a unitary transformation on c. (Any 2×2 unitary operator can be used here, but Hadamard gates are commonly used for this task.)

$$|n\rangle|c\rangle \to |n\rangle H|c\rangle \tag{4.12}$$

Then we apply a 'shift' (S) to move either to the left or to the right depending on the value of the "flipped coin".

$$S|n,0\rangle \to |n-1,0\rangle \tag{4.13}$$

$$S|n,1\rangle \to |n+1,1\rangle \tag{4.14}$$

If we apply t steps on the initial state $|i,0\rangle$, we reach a state $(SH)^t|i,0\rangle$. These are of course implemented in quantum states and not in classical ones. Hence, the statistics we obtain are much more complicated than the ones in the real analogue. However if at each step, we first measure the value of the "flipped coin" before applying the 'shift', then the overall process collapses exactly to the discrete time classical random walk.

Note that the infinite dimensional straight line graph is a regular one where the degree for each node (and hence the possible directions to take at each step) is two. This is why a 2-state coin is used in the above process. Using similar methods, a d-state coin can define the discrete time random walk on a d-regular graph. In this case, for all vertices, each of the d outgoing edges should be uniquely labeled by one of the d base states of the coin register so that the values of the flipped coin at each step can determine a direction to take.

The above process is not yet applicable to the walks on a non-regular graph such as the finite line in Figure 4.1. There are several alternatives to make it useful also for non-regular

graphs. One way is to use a conditional coin operator. In this case, the position on the graph which is coded into the register *n* will be used to choose one among the alternative coin operators for vertices of varying degrees. Another way is to add self loops to the graph to make it regular. In this case, the transitions on the self loops will simply keep the walk where it is. There may be other ways of going around this problem. No matter which is chosen, it is possible to define and efficiently implement the corresponding quantum random walk for any classically defined random walk.

4.2.3. Quantum Random Walk on a Hypercube

We have examined how to define random walks on quantum computers and while doing this we have extensively used the straight line graph as a trivial example. For a less trivial example, let us now consider quantum walks on a graph with different characteristics, the hypercube. In the following parts, we will first give a formal definition for a hypercube, then consider the continuous and discrete random walks on a hypercube of three dimensions.

A convenient way to define an *n*-dimensional hypercube graph, $G = (V(G), E(G))$, is through binary labeling of its vertices and Hamming distances between the labels. (The Hamming distance *X*-*Y* between two strings *X* and *Y* is defined to be the number of bits in which these strings differ.) Hence, we use *n*-bit string to label the members of $V(G)$ and allow edges only between those pairs of vertices whose labels are separated by a Hamming distance of 1. Then we have the set of vertices defined by $V(G) = \{(x_0 x_1 .. x_{n-1}) \mid x_i \in \{0,1\} \text{ for } 0 \le i \le n-1\}$ and the set of edges defined by $E(G) = \{(X, Y) \mid X, Y \in V(G) \wedge X - Y = 1\}$. Figure 4.2 depicts a 3-dimensional hypercube which is structured in this way.

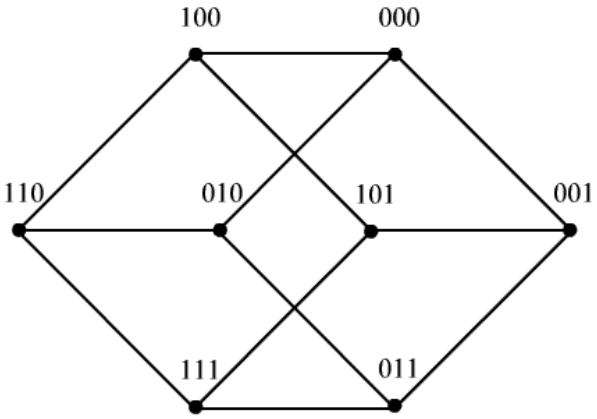

Figure 4.2. The three dimensional hypercube

The continuous time quantum random walk on $G = (V(G), E(G))$ can be defined in the usual way. Note that G is regular so the choices of $H_1 = \gamma L$ or $H_2 = \gamma A$ as the Hamiltonian will be equivalent since they differ by a multiple of the identity operator. Let us use $H = \gamma A$ as the Hamiltonian and let $\gamma = 1$. Then H is given by

$$H = \gamma A = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \end{bmatrix} \tag{4.15}$$

Hence running the quantum walk on the 3-dimensional hypercube for duration Δt is simply equivalent to simulating H for duration $\Delta t : |\psi(t + \Delta t)\rangle = e^{-iH\Delta t}|\psi(t)\rangle$. More generally, the adjacency matrix for the n-dimensional hypercube is given by

$$A = \sum_{j=1}^{n} X_j \tag{4.16}$$

where X_j is the operator which acts as the Pauli X on the j'th qubit while leaving other qubits untouched. Then, the continuous random walk on a general n-dimensional hypercube is defined by the Hamiltonian $H = \sum_{j=1}^{n} X_j$, which is a sum of local terms and therefore it can be simulated efficiently.

If we consider the discrete time random walk on G, we first note that G is regular, hence we can define the discrete time random walk on G according to the coin flip model introduced in 4.2.2 without any modifications. Note that G is 3-regular, and hence we need a coin of three states. As stated before, any unitary matrix of the correct size can serve as the coin operator. However, the choice of the coin operator determines the characteristics of the quantum walk and hence is an important parameter in specifying the desired effects. Hadamard gates, for example, can be used as coin operators to specify an unbiased walk. Another commonly used coin operator is known as Grover's diffusion (or Grover's coin) operator which is defined on n states by the general term: $D_{i,j} = \frac{2}{n} - \delta_{i,j}$. This operator is a member of the family of permutation symmetric operators, which carry interesting features and in fact, Grover's coin is the one among these, which is farthest away from the identity operator.

By manipulating the coin flip operator we can define the discrete time quantum walk in a flexible manner. This flexibility can be viewed as an advantage over the continuous time quantum walk, where the only manageable parameter is the local features of the Hamiltonian. Let us make use of this flexibility and use a 4-state coin to describe the discrete time quantum walk on the graph G.

The state space for the discrete time walk on G is spanned by the state of the form $|x_2 x_1 x_0, c_1 c_0\rangle$ where the first register $x_2 x_1 x_0$ defines the position on the graph by specifying the label of the current vertex and the second register $c_1 c_0$ is the two-bit register for representing the coin. We employ two-bit Hadamard operator as the coin flip operator ($C = I \otimes I \otimes I \otimes H \otimes H$) and associate one of the four outcomes of the coin flip with a 'do

nothing' action in the 'shift' phase, and the other three will be associated with the possible motions at each step. But first, we need to label the edges of the graph to indicate directions. A simple rule will guide us in doing so: an edge is labeled by *r* if the labels for the vertices incident to this edge differs in the *r*'th bit. (We number bits from right to left and start with 0.) Consider the 3-dimensional hypercube in Figure 4.3.

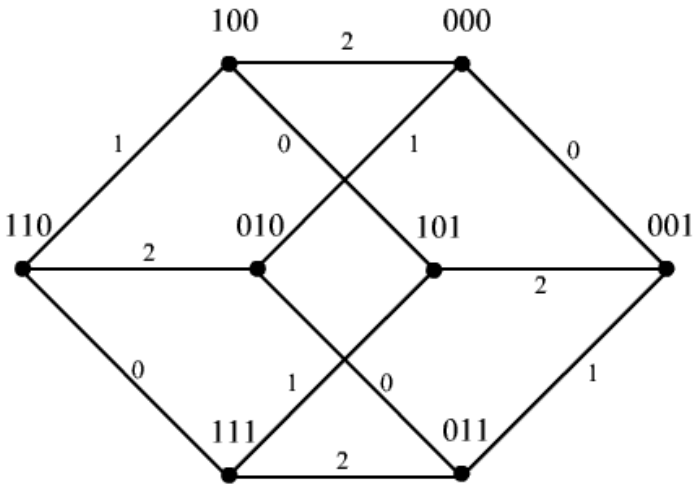

Figure 4.3. The 3-dimensional hypercube with labeled edges

Then the shift operator acts on the base states as follows

$$S\left|x_2x_1x_0,00\right\rangle \rightarrow \left|x_2x_1\bar{x}_0,00\right\rangle \tag{4.17}$$

$$S\left|x_2x_1x_0,01\right\rangle \rightarrow \left|x_2\bar{x}_1x_0,01\right\rangle \tag{4.18}$$

$$S\left|x_2x_1x_0,10\right\rangle \rightarrow \left|\bar{x}_2x_1x_0,10\right\rangle \tag{4.19}$$

$$S\left|x_2x_1x_0,11\right\rangle \rightarrow \left|x_2x_1x_0,11\right\rangle \tag{4.20}$$

where the complement operation is defined as $\bar{0}=1$ and $\bar{1}=0$. Note that the shift operation is defined so that if the coin is in state $\left|3\right\rangle\left(=\left|11\right\rangle\right)$ then we take no action and if the coin is in a base state $\left|r\right\rangle$ other than $\left|3\right\rangle$, we follow the edge incident to current vertex and labeled by *r*, which is equivalent to taking the complement of the *r*'th bit in the position register.

Finally we can define the state after t steps in the usual way as $(SC)^t|000,00\rangle$, where we assumed the initial state $|000,00\rangle$.

4.2.4. The Relation Between Discrete and Continuous Time Quantum Walks

In the classical case, a continuous time random walk is the limit case for a discrete time random walk where the time spacing between the discrete steps approaches to zero. The relation between the quantum versions is not so clear. It is yet an open question whether the models for discrete and continuous time quantum walks can be derived from each other. The difficulty is mainly due to the use of extra registers (or coins) in the discrete time quantum walk, while these are not needed in the continuous time quantum walk.

Although the two models seem to be quite different, the results obtained by use of them are similar in efficiency and computational cost. This has been observed by several researchers in different contexts [39, 39]. In [42], some statistical measures of performance like the quantum counterpart of mixing time and absorption probability are stated to be the same for the two models. These similarities are sometimes viewed as evidence for the equivalence of the two models in computational power, however there are also results which point out the algorithmic differences between the discrete and continuous walks [46].

4.3. Algorithmic Use of Quantum Walks

Quantum walks are widely known to have different properties than their classical counterparts, mostly due to the nature of quantum interference which can act in a destructive manner unlike in the classical case. An important observation is that unlike their classical counterparts, quantum walks do not converge to a stationary state but instead a time average of the state of the walk does converge to a limit [41]. It is on this convergence that one can define a mixing time for quantum walks, and it is known that quantum walks for many graphs have quadratically smaller mixing times when compared to their classical counterparts.

A more interesting difference between classical and quantum walks is that of hitting times. It can be shown that quantum walks have exponentially smaller hitting times on some graphs when compared to classical walks. This exponential gain against the classical walks is the inspiration for some algorithms depending on quantum walks. Such an example can be found in [47], where quantum random walks were shown to be exponentially faster than classical walks for the task of finding the exit node starting from the entrance node of the sort of graphs shown in Figure 4.4. (A similar problem, where the structure of the graph is modified, will be examined in detail in Chapter 5.)

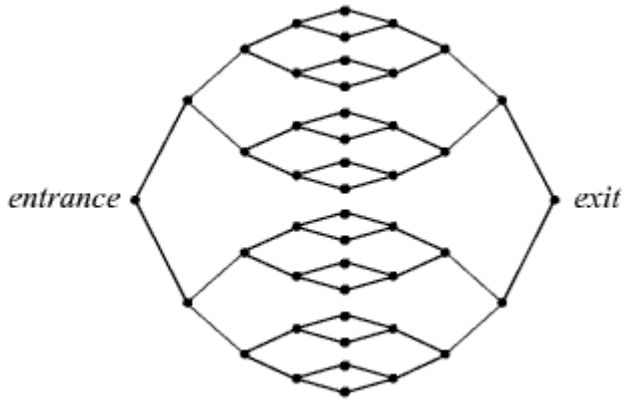

Figure 4.4. The graph constructed by unifying the leafs of two binary trees of same size

A set of similar results are also demonsrated in [43]. However it should be noted that what we are talking about is not yet an exponential algorithmic gain since there are classical algorithms other than random walks, against which the gain of quantum walks is only polynomial in solving the mentioned problems.

Quantum walk search algorithms form another interesting group of quantum algorithms depending on quantum walks. In the general scenario, the walk takes place on a graph, some vertices of which are marked. The task is to find such a vertex. At each step, the walk either proceeds to an adjacent vertex or it queries a black box to see if the current vertex is a marked one. When the search problem is formulated in this way, Grover's algorithm runs in $O\left(M\sqrt{N}\right)$ steps, where N is the number of vertices in the graph and M is the maximum distance between two vertices of the graph.

The hypercube graphs introduced in the previous section are particularly useful in studying quantum search algorithms. In an *n*-dimensional hypercube graph, the maximum distance between two vertices is *n* and the number of vertices is $N = 2^n$. Therefore Grover's algorithm can solve the search problem on the *n* dimensional hypercube in $O\left(\log N\sqrt{N}\right)$ steps while it is shown in [48] that quantum walk search can do the same in $O\left(\sqrt{N}\right)$ steps.

The quantum walk search on an *n*-dimensional grid of *N* vertices is another interesting application. It is studied by use of continuous time quantum walks in [49] and by use of discrete time quantum walks in [46]. The results obtained by use of discrete time quantum walks seem to be dependent on the choice of the coin operator and for various values of *n,* they are better than those obtained by use of continuous time quantum walk.

Quantum walks on various graphs exhibit interesting behavior and thus the algorithms employing them can come up with striking results. The exponential algorithmic speedup by a quantum walk is one such result and it will be discussed in Chapter 5.

5. A FAST ALGORITHM EMPLOYING THE QUANTUM RANDOM WALK

Determining the problems which quantum computers can solve qualitatively faster than classical computers and designing quantum algorithms which exhibit this speedup against classical algorithms, is a central issue in the field of quantum computation. A polynomial speedup over the best classical algorithms for a task is almost always considered as valuable, however it is exponential speedup that is thought to fully demonstrate the power of quantum computation. Until very recently, only those quantum algorithms which are essentially based on the quantum Fourier transformation had been shown to provide exponential speedup over their classical counterparts, and these algorithms could only solve several variants of hidden subgroup problems [25].

Meanwhile, quantum random walks were widely known to be superior to classical random walks in various degrees for various problems, however, there was no result demonstrating the advantage of quantum random walks over the general class of classical algorithms. The work by Childs *et al* showed that an algorithm based on the continuous time quantum random walk can provide exponential speedup over the best classical algorithm for the solution of a black box problem defined on some special graph [3]. (It was later claimed that the discrete time quantum walk could serve as well for the same task. [50]) This result is important in several ways. The quantum algorithm introduced in [3] is in no way similar to the ones depending on the quantum Fourier transformation technique and yet it is able to provide exponential speedup over the best possible classical algorithm. This encourages researchers to search for other quantum computation techniques that can provide exponential speedup. Another point is that we can expect this new technique (or some variants of it) to be used also in speeding up the solutions of some real life problems on quantum computers. This expectation is primarily based on the widespread use of random walk algorithms in classical computation.

In the following parts we will examine, in detail, how Childs *et al* demonstrated exponential speedup by quantum random walks. First, we will provide a definition of their problem in Section 5.1. Then in Section 5.2 we will briefly introduce the algorithm they proposed. Section 5.3 will aim to show how to implement the quantum random walk efficiently on an arbitrary graph with the help of the black boxes. Section 5.4 provides an analysis of the performance of the proposed method in terms of time requirements and probability of success. Finally, in Section 5.5, we will discuss why classical algorithms can not solve the same problem effectively in subexponential time.

5.1. The Graph Traversal Problem

The problem we are dealing with, is a black box graph traversal problem defined on the graphs G_n of a specific form, where n is a positive integer. G_n consists of two binary trees of height n, connected by a random cycle that alternates between the leaves of the two trees. The cycle is arranged in a way that every leaf of a tree is connected to two of the leaves of the other tree. Figure 5.1 contains a typical graph of the form G_4. (The graphs G_n and some variants of them are sometimes called "glued trees". We will use this term only for the sort of graphs shown in Figure 5.1.)

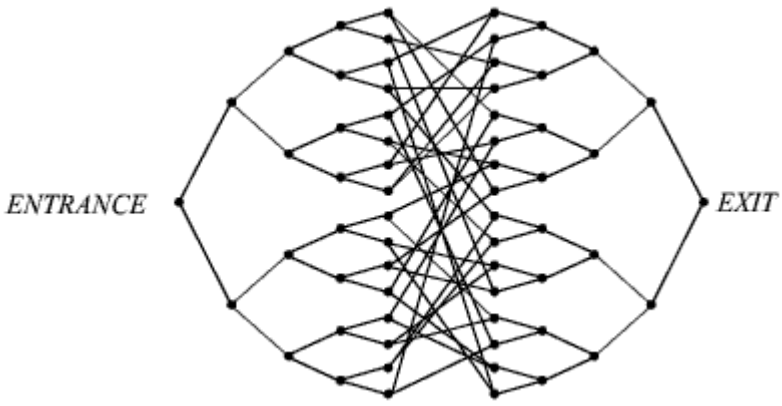

Figure 5.1. An instance of G_4, the glued trees of height 4

The problem assumes the existence of a black box, the functionality of which would be better understood after the following definitions. Let $G = (V(G), E(G))$ be an undirected graph with *N* vertices.

Each vertex $u \in V(G)$ is assigned a distinct *m*-bit string as its name where $2^m > N$. In particular, the string, $1^m = 11..1$, is not assigned as name of any vertex. (The reason will be clear soon.) For each vertex, $u \in V(G)$, the outgoing edges of *u* are assigned labels from a set *L* of size *k,* where *k* is at least as large as the maximum vertex degree in *G*.

Next, we define the function $v_c(u)$ for $u \in \{0,1\}^m$ and $c \in L$ as follows: If $u \in V(G)$ and if u has an outgoing edge labeled by c then $v_c(u)$ is the name of the vertex reached by following the outgoing edge of u labeled by c. If $u \notin V(G)$ or if u does not have an outgoing edge labeled by *c*, then $v_c(u) = 1^m$.

We define a black box V for computing $v_c(u)$ on the input $c \in L$ and $u \in \{0,1\}^m$. The functionality of V can be formulated as a unitary transformation as in (5.1), where *a* and *b* are *m*-bit registers.

$$V|c,a,b\rangle = |c,a,b \oplus v_c(a)\rangle \tag{5.1}$$

In the following parts, we will sometimes combine the black box V and its first input $c \in L$ to form a component V_c of the black box for answering the queries whose first input is *c*. Then for each $c \in L$ we define V_c as a circuit component that computes $v_c(u)$ on input $u \in \{0,1\}^m$.

$$V_c|a,b\rangle = |a,b \oplus v_c(a)\rangle \tag{5.2}$$

Finally, a formal definition of the graph traversal problem is as follows:

Let G be a graph and *ENTRANCE* and *EXIT* be two vertices of G. The input of the traversal problem is a black box for G and the name of the *ENTRANCE*. The output is the name of the *EXIT*.

In the following, we will be dealing with the graph traversal problem on the graphs of the form G_n. Then the task is to develop an efficient mechanism which, when given the name of the *ENTRANCE* node of a graph of the form G_n, can manage to find out the name of the *EXIT* node by querying the black boxes.

5.2. The Algorithm for the Graph Traversal Problem on the Glued Trees

In [3], it is shown that the instances of the graph traversal problem on the special graphs of the form G_n, can be efficiently solved to any accuracy by a quantum algorithm. The algorithm depends primarily on running a continuous time quantum random walk on G_n. It roughly consists of the following steps.

Inputs: A black box for an undirected graph G_n of the special form introduced in Section 5.1 and a positive value for acceptable accuracy.

Step 0: Determine an interval I according to the size of the graph, the desired accuracy of results and some statistical measures of the structure of the graph.

Step 1: Pick a random value t from the interval I.

Step 2: Run the continuous time quantum random walk on G_n (with the help of the black box), for duration t, starting from the ENTRANCE node.

Step 3: After duration t perform a measurement to get the name of a vertex in the graph.
Check if this is the name of the EXIT node. If yes, return this name and terminate. If not, go to step 1.

The algorithm depends on running the continuous time quantum random walk on the graph G_n for a duration randomly chosen from a predefined interval. This step is repeated until the desired result is obtained from a measurement, which is performed at the end of each separate run. This is just a sketch of the algorithm. The details of each step will be examined soon, but we will not be doing this in the order specified above. We should begin with describing how to implement the continuous time quantum random walk on the graph G_n. The reasoning behind the other steps comes after this.

5.3. The Quantum Walk with a Black Box

In this section, we will examine how to implement the quantum walk on a graph $G = (V(G), E(G))$, with the help of the black box V. We have examined the continuous time quantum random walk model in detail, in Chapter 4. So we can now apply it.

As we had seen, running the continuous time random walk on a graph $G = (V(G), E(G))$, for duration t is simply equivalent to simulating a Hamiltonian H for that duration where H reflects the local properties of the graph. The adjacency matrix of the graph whenever it is available serves well for this task. (We simply let $\gamma = 1$.)

$$H_{ab} = \begin{cases} 1 & (a,b) \in E(G) \\ 0 & otherwise \end{cases} \tag{5.3}$$

The problem introduced in Section 5.1 does not give information about the internal structure of the graph but it provides access to a black box instead. This will be helpful in

simulating e^{-iHt}, which is equivalent to running the continuous time random walk on the graph $G = (V(G), E(G))$, for duration t.

At this point, we need to make a separation between the graphs whose edges are symmetrically labeled and the others. A symmetrically labeled graph is one where $v_c(a) = b$ implies $v_c(b) = a$. (Hence, note that $v_c(v_c(a)) = a$ for a symmetrically labeled graph.) We will first show that for a symmetrically labeled graph, e^{-iHt} can be simulated efficiently with the help of black box queries. Then we will generalize this result to the graphs which are not symmetrically labeled. We have the following theorem:

Theorem 5.1. *Given the black box for a symmetrically labeled graph G, a continuous time quantum walk on G can be simulated for time t with precision* ε *by using* $O(k^2 t^2 / \varepsilon)$ *black box queries and* $O(k^2 t^2 m / \varepsilon)$ *auxiliary operations.*

The Hilbert space for the quantum walk on *G* is spanned by states of the form $|a, b, r\rangle$ where a and b are *m*-bit strings and r is a bit. The states of the form $|u,0,0\rangle$ correspond to the vertices $u \in V(G)$.

Since we do not have direct access to the structure of the graph (and hence to the adjacency matrix), and can reach it only through access to a black box, we need to redefine the Hamiltonian *H* of the system without referring to the structure of the graph, but with the freedom to use the black box queries.

First, note that, for all $a \in V(G)$, *H* maps a to an equal superposition of all neighbors of a. In other words *H* maps $|a,0,0\rangle$ to $\sum_{c: v_c(a) \in V(G)} |v_c(a),0,0\rangle$. Then it may be useful to write,

$$H = \sum_{c \in L} H_c \tag{5.4}$$

where for each $c \in L$, H_c acts on the state $|a,0,0\rangle$ as in (5.5)

$$H_c|a,0,0\rangle = \begin{cases} |v_c(a),0,0\rangle & v_c(a) \in V(G) \\ 0 & \textit{otherwise} \end{cases} \tag{5.5}$$

The linear combination rule (which we mentioned in Section 3.2.6) suggests that if we could efficiently simulate the individual terms, H_c, in the summation (5.4), then it would also be possible to efficiently simulate their combination *H*. But each individual term H_c has the same form shown in (5.5). So the problem is reduced to simulating this simpler process in which we make use of the black box queries and an additional operator *T*, to be explained shortly.

The black box queries enable us to access the structure of the graph. For a vertex a and a label c, we can extract $v_c(a)$, the name of the vertex on the other side of the edge c, by querying the black box component V_c on input a. (Recall that this is equivalent to querying the black box V on the inputs c and a.) However, we also want to know if the result is a name of a vertex or 11..1, which is the case when $a \notin V(G)$ or there is no edge labeled by c adjacent to a. So we make use of the third register in the general state vector $|a,b,r\rangle$ for this task. Then the black box queries are represented unitarily as

$$V_c|a,b,r\rangle = |a, b \oplus v_c(a), r \oplus f_c(a)\rangle \tag{5.6}$$

where the function f_c is defined by

$$f_c(a) = \begin{cases} 0 & v_c(a) \in V(G) \\ 1 & \textit{otherwise} \end{cases} \tag{5.7}$$

Now since $v_c(v_c(a)) = a$ for a symmetrically labeled graph, we have

$$V_c\left(V_c\left|a,0,0\right\rangle\right)=V_c\left|a,v_c(a),f_c(a)\right\rangle=\left|a,v_c(a)\oplus v_c(a),f_c(a)\oplus f_c(a)\right\rangle=\left|a,0,0\right\rangle, \tag{5.8}$$

which suggests

$$V_c^{\dagger}=V_c^{-1}=V_c. \tag{5.9}$$

We need to introduce one more operator, T, whose action is defined by $T\left|a,b,0\right\rangle=\left|b,a,0\right\rangle$ and $T\left|a,b,1\right\rangle=0$. This can be written in a more compact form as

$$T\left|a,b,r\right\rangle=\delta_{0,r}\left|b,a,0\right\rangle \tag{5.10}$$

where $\delta_{0,r}=1$ if $r=0$ and $\delta_{0,r}=0$ otherwise. Let us now see what V_c and T can do together, when they are combined in the form $V_c^{\dagger}TV_c$.

$$\begin{aligned} V_c^{\dagger}TV_c\left|a,0,0\right\rangle &= V_c^{\dagger}T\left|a,v_c(a),f_c(a)\right\rangle \\ &= \delta_{0,f_c(a)}\,V_c^{\dagger}\left|v_c(a),a,0\right\rangle \\ &= \delta_{0,f_c(a)}\left|v_c(a),a\oplus v_c(v_c(a)),f_c(v_c(a))\right\rangle \\ &= \delta_{0,f_c(a)}\left|v_c(a),0,0\right\rangle \end{aligned} \tag{5.11}$$

We can translate this as follows: For each $c\in L$, $H_c=V_c^{\dagger}TV_c$.

$$H_c\left|a,0,0\right\rangle=V_c^{\dagger}TV_c\left|a,0,0\right\rangle=\begin{cases}\left|v_c(a),0,0\right\rangle & f_c(a)=0\ \left(v_c(a)\in V(G)\right)\\ 0 & f_c(a)=1\ \left(v_c(a)\notin V(G)\right)\end{cases} \tag{5.12}$$

We have been looking for an efficient simulation of H_c for each $c\in L$ and we have noted that $H_c=V_c^{\dagger}TV_c$. Since we are provided with the black box components $V_c^{\dagger}=V_c$, in order to conclude that H_c can be simulated efficiently for each $c\in L$, we only need to show

that the operator T can be simulated efficiently. We will be doing this in the following parts. First, note that T can be formulated as

$$T = S^{(1,\,m+1)} \otimes S^{(2,\,m+2)} \otimes \ldots \otimes S^{(m,\,2m)} \otimes |0\rangle\langle 0| \tag{5.13}$$

where $S^{(i,\,j)}$ is the two-bit swap operator that acts on the i'th and j'th bits and $|0\rangle\langle 0|$ (the projector onto $|0\rangle$) acts on the third register. In the computational basis, if the third register's content is $|0\rangle$ the swap operators exchange the content of the first two registers, otherwise if the third register's content is $|1\rangle$, the whole state is reduced to 0, which is the amplitude of the projection of the third register on $|0\rangle$. The two bit swap operator S is given by

$$S = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.14}$$

Recall that in Section 3.3.2 we had developed a model for the simulation of Hamiltonians with non-local terms. We will extend this model for the simulation of T. Let us begin with examining the diagonal form of the two-bit swap operator S, obtained by spectral decomposition. (Note that all unitary operators U, are normal: $UU^{\dagger} = U^{\dagger}U$. Hence spectral decomposition can be applied to them.)

$$S = W^{\dagger}\Lambda W = \begin{bmatrix} \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} & 0 \\ 0 & \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \\ 0 & \frac{1}{\sqrt{2}} & 0 & \frac{-1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & 0 & \frac{-1}{\sqrt{2}} & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} & 0 & 0 & \frac{1}{\sqrt{2}} \\ 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{2}} & 0 & 0 & \frac{-1}{\sqrt{2}} \\ 0 & \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} & 0 \end{bmatrix} \tag{5.15}$$

The diagonal entries of Λ are the eigenvalues of S and the columns of the unitary matrix $W^\dagger$ are the corresponding eigenvectors. This picture suggests that if we could first perform W, then simulate Λ for a time Δt, and then perform $W^\dagger$, the overall effect of these operations would be to simulate S for a time Δt. W and $W^\dagger$ are two qubit unitary operators, so they can be approximated to arbitrary accuracy. In between the two, there is the need for simulating the operator Λ, which acts on the computational basis according to

$$\Lambda|00\rangle = |00\rangle, \tag{5.16}$$

$$\Lambda|01\rangle = |01\rangle, \tag{5.17}$$

$$\Lambda|10\rangle = |10\rangle, \tag{5.18}$$

$$\Lambda|11\rangle = -|11\rangle. \tag{5.19}$$

Then speaking in the computational basis we can say, if the input state is $|11\rangle$, Λ applies a phase shift of -1 to this state and if the input state is not $|11\rangle$ then Λ leaves it unchanged. Note the similarity between Λ and the one-bit operation *Z* which applies a phase shift of -1 to its input if it was initially at state $|1\rangle$. We will make use of this similarity to modify the circuit for simulating Hamiltonian $Z \otimes Z \otimes .. \otimes Z$ (see Section 3.3.2 for the details) to construct a circuit which simulates the Hamiltonian $\Lambda \otimes \Lambda \otimes .. \otimes \Lambda$. The thing to do is to use Toffoli gates to count the parity of the total number of qubit pairs in the state $|11\rangle$ instead of using CNOT gates to count the parity of the total number of single qubits in the state $|1\rangle$. The circuit shown in Figure 5.2 implements this idea to simulate the Hamiltonian $\Lambda \otimes \Lambda \otimes .. \otimes \Lambda$ for time Δt.

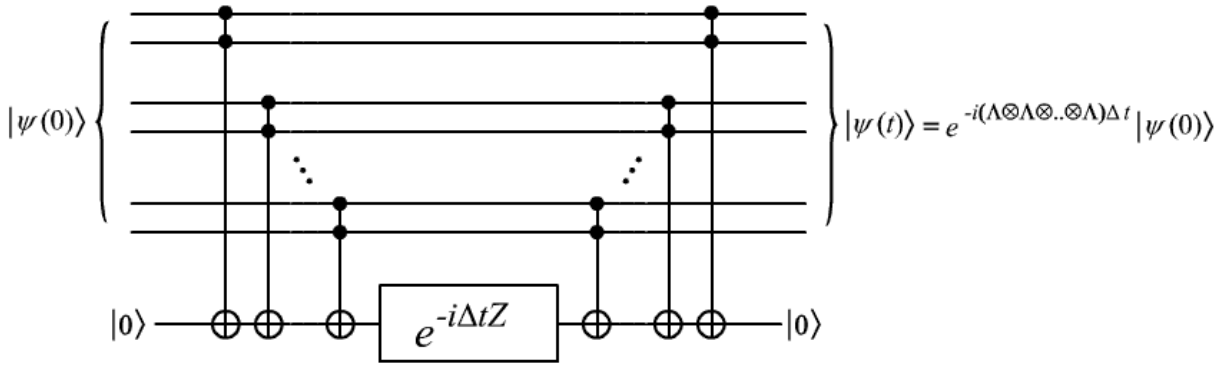

Figure 5.2. The circuit for simulating the Hamiltonian $\Lambda \otimes \Lambda \otimes .. \otimes \Lambda$ for duration Δt

We can transform the circuit in Figure 5.2 to one that simulates $S \otimes S \otimes .. \otimes S$ by adding the operators W and $W^\dagger$ into the picture in the appropriate way. Then, extending the state space by one qubit on which the Hamiltonian $|0\rangle\langle 0|$ acts, we can form a circuit for simulating T. The last qubit, on which the Hamiltonian $|0\rangle\langle 0|$ acts, is effectively a control bit on the evolution $e^{-i(S\otimes S\otimes..\otimes S)t}$. Speaking in the computational basis, if the state of the last qubit is $|0\rangle$, the evolution $e^{-i(S\otimes S\otimes..\otimes S)t}$ is carried over the other qubits, and if the state of the last qubit is $|1\rangle$, no change is applied to the overall input state. Then the circuit for simulating T for duration Δt can be constructed as in Figure 5.3.

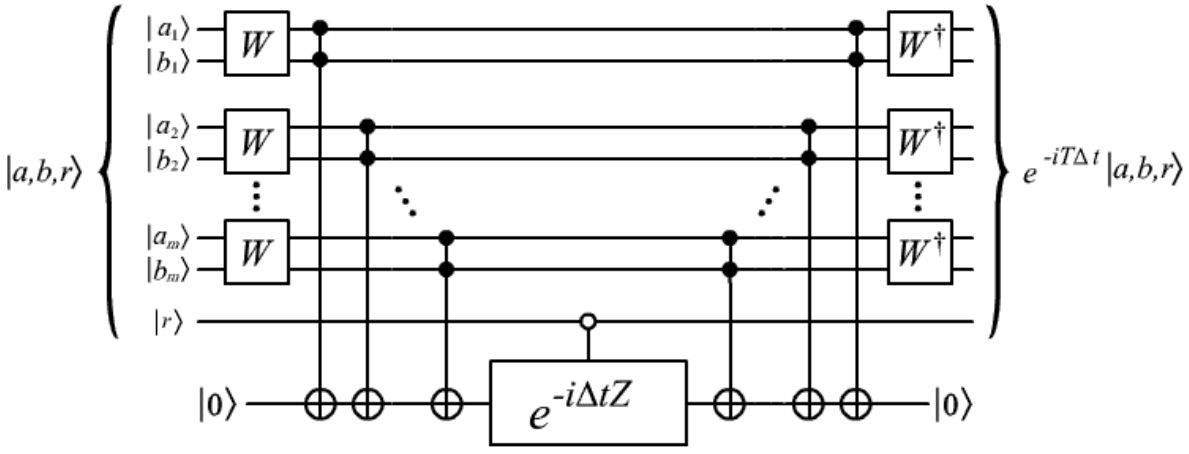

Figure 5.3. The circuit for simulating the Hamiltonian T for duration Δt

The circuit in Figure 5.3 can simulate $e^{-iT\Delta t}$ for arbitrary Δt, using only $O(m)$ one and two qubit gates. Then, as a result of the unitary conjugation rule, we can efficiently simulate $V_c^\dagger T V_c$ for $c \in L$. Finally, by the linear combination rule, $\sum_c V_c^\dagger T V_c$ can be simulated efficiently.

For the sort of graphs we are dealing with, the number of H_i, H_j pairs in the error term of (3.12) which fail to commute is in the order of $O(q)$. Therefore we can establish a tighter bound for the error term in Lie product formula (See equation (3.15).) which is $O(q\ell t^2/r)$. Hence, $\sum_c V_c^\dagger T V_c$ can be simulated with precision ε in $r \in O(kt^2/\varepsilon)$ iterations, each with k runs of $V_c^\dagger T V_c$ for k distinct $c \in L$. Then, the overall complexity of simulating $H = \sum_c V_c^\dagger T V_c$ for duration t with precision ε is $O(k^2 t^2/\varepsilon)$ black box queries and $O(k^2 t^2 m/\varepsilon)$ auxiliary operations. This completes the proof of Theorem 5.1. Next, we will generalize this result to the case where symmetric labeling is not required.

Theorem 5.2. *If G is bipartite and $a \in V(G)$ then, given any black box for G, a quantum walk on G starting in state $|a\rangle$ can be simulated for time t with precision ε by using $O(k^4 t^2/\varepsilon)$ queries and $O(k^4 t^2 m/\varepsilon)$ auxiliary operations.*

To prove Theorem 5.2, we need to transform the case of a general graph to the case of a symmetrically labeled graph and then achieve the desired result by employing Theorem 5.1.

Recall that for a symmetrically labeled black box we had decomposed the Hamiltonian H into the constituents H_c for $c \in L$. Each of these constituents could be written as $H_c = V_c^\dagger T V_c$ depending on the fact that $v_c(v_c(a)) = a$ for symmetric labeling. An asymmetrically labeled black box does not necessarily satisfy $v_c(v_c(a)) = a$. Hence we need another way of defining

the constituent Hamiltonians which build up the global Hamiltonian H for the case of asymmetric labeling.

In asymmetric labeling, each edge $(a,b) \in E(G)$ is associated with two labels $c_a \in L$ and $c_b \in L$, one from each of the two endpoints of the edge. (The symmetric labeling can be viewed as a special case of this, where c_a and c_b have to be same.) To simplify things, we desire a single label for each edge $(a,b) \in E(G)$. This can be done by putting the two labels c_a and c_b together as (c_a, c_b) or (c_b, c_a) to form a unique label $(c_i, c_j) \in L \times L$ of the edge. Note however that, this is not the way the black box codes the graph and there is the need for implementing our view in terms of the black box's functionality.

We begin with redefining the Hamiltonian H as

$$H = \sum_{(c_i,c_j)\in L\times L} H_{(c_i,c_j)}, \tag{5.20}$$

where we require that each constituent should satisfy (5.21).

$$H_{(c_i,c_j)} |a,0,0\rangle = \begin{cases} |v_{c_i}(a),0,0\rangle & v_{c_j}(v_{c_i}(a)) = a \\ 0 & \text{otherwise} \end{cases} \tag{5.21}$$

Next, we define the function $f_{(c_i,c_j)}(a)$ as in (5.22). Note that computing $f_{(c_i,c_j)}(a)$ requires two queries of the asymmetric black box and $O(m)$ additional one and two qubit gates in the simplest construction.

$$f_{(c_i,c_j)}(a) = \begin{cases} 0 & v_{c_j}(v_{c_i}(a)) = a \\ & \\ 1 & \text{otherwise} \end{cases} \tag{5.22}$$

We define the component $V_{(c_i,c_j)}$ as

$$V_{(c_i,c_j)}\left|a,b,r\right\rangle = \left|a,b\oplus v_{c_i}(a),r\oplus f_{(c_i,c_j)}(a)\right\rangle \tag{5.23}$$

Implementing $V_{(c_i,c_j)}$ requires three queries of the asymmetric black box, two for computing $f_{(c_j,c_i)}(a)$ and one for the usual task of computing $v_{c_i}(a)$. $O(m)$ additional one and two qubit gates are also needed. Finally let us examine the functionality represented by $V_{(c_j,c_i)}TV_{(c_i,c_j)}$ where the definition for T is the same as before.

$$\begin{aligned} V_{(c_j,c_i)}TV_{(c_i,c_j)}\left|a,0,0\right\rangle &= V_{(c_j,c_i)}T\left|a,v_{c_i}(a),f_{(c_i,c_j)}(a)\right\rangle \\ &= \delta_{0,f_{(c_i,c_j)}(a)}\, V_{(c_j,c_i)}\left|v_{c_i}(a),a,0\right\rangle \\ &= \delta_{0,f_{(c_i,c_j)}(a)}\left|v_{c_i}(a),a\oplus v_{c_j}\left(v_{c_i}(a)\right),f_{(c_j,c_i)}\left(v_{c_i}(a)\right)\right\rangle \\ &= \delta_{0,f_{(c_i,c_j)}(a)}\left|v_{c_i}(a),0,0\right\rangle \end{aligned} \tag{5.24}$$

Hence, we have

$$H_{(c_i,c_j)} = V_{(c_j,c_i)}TV_{(c_i,c_j)}, \tag{5.25}$$

$$H = \sum_{(c_i,c_j)\in L\times L} V_{(c_j,c_i)}TV_{(c_i,c_j)}\,. \tag{5.26}$$

This completes our construction of a model which can mimic the model for symmetric labeling case. Instead of k labels in the set L, we now have k^2 labels in the set $L\times L$. So by Theorem 5.1 the quantum walk on G can be simulated for time t, with precision ε by using $O\left((k^2)^2t^2/\varepsilon\right)$ queries and $O\left((k^2)^2t^2m/\varepsilon\right)$ auxiliary operations. This concludes the proof of Theorem 5.2.

By proving Theorem 5.1 and Theorem 5.2, we have shown that a continuous time quantum random walk can be implemented efficiently on an arbitrary undirected graph as long as the access to a black box (like the one specified in Section 5.1) for that graph is provided. For this result to be useful in the solution of the graph traversal problem, we need to analyze how fast the quantum walk traverses the glued trees graph G_n.

5.4. Upper Bound on the Traversal Time

In this section we will consider the quantum walk on the specific family of graphs G_n we had introduced in Section 5.1. We are going to try to prove that the quantum walk on this family of graphs reaches the *EXIT* node within polynomial time.

5.4.1. The Quantum Walk on the Column Subspace

The state vector describing the quantum walk on a graph lies in an N dimensional Hilbert space where N is the number of vertices in the graph. The sort of graphs we are dealing with have the nice property that we can move to a smaller space where the analysis would be simpler. This space is known as the column subspace and it will be introduced in this section. First, we should clarify what we mean by a column.

A vertex $u \in V(G)$ is said to be on *column j* of graph $G_n = (V(G), E(G))$, if the shortest path from this vertex to the *ENTRANCE* vertex has length j. We prefer the name "column" since those vertices which are in the same column are depicted as vertically aligned in Figure 5.1, which we used while introducing the type of graphs we are dealing with.

The column subspace is a $2n+2$ dimensional subspace spanned by the states

$$\left|C_j\right\rangle = \frac{1}{\sqrt{N_j}} \sum_{a \in column\ j} \left|a\right\rangle, \tag{5.27}$$

where N_j is the number of vertices in column *j*. It should be noted that $\left|C_j\right\rangle$ is the unit ket vector which lies in the same direction as the sum of the *N* dimensional unit ket vectors $\left|a\right\rangle,\left|b\right\rangle,..$ standing for the vertices on column *j*. Therefore, a general state of the quantum walk can be represented in the column subspace if and only if the vertices sharing the same column also share the same amplitude in the *N* dimensional general state vector. Only these states can be represented as a linear combination of the vectors $\left|C_j\right\rangle$ for $0 \le j < 2n+2$. Let

$$\left|C\right\rangle = \sum_{j=1}^{2n+2} A_j \left|C_j\right\rangle. \tag{5.28}$$

The amplitude of $\left|C_j\right\rangle$ in the general state vector is A_j, if all the vertices in column *j* have individual amplitudes of $A_j \big/ \sqrt{N_j}$. These constraints are not restrictive in analyzing the quantum walk on the glued trees, because the column subspace is invariant under the quantum walk (i.e. a quantum walk starting from a state in the column subspace remains in the column subspace) and we begin the walk from a state ($\left|ENTRANCE\right\rangle$) which is in the column subspace. The invariance is a result of the structure of the graph, which guarantees uniform distribution of amplitudes between the members of the same column.

If we are to analyze the quantum walk on the column subspace, we need to examine the structure of the transitions available in this new space. We should determine the pair of columns $\left|C_i\right\rangle$, $\left|C_j\right\rangle$ which are linked in the sense that a transition among them is possible. We should also determine the relative strength of these links. This analysis will give us a new graph on which we will analyze the quantum walk.

It is trivial to state that a transition is possible only between the adjacent columns since only those vertices in the adjacent columns are linked to each other in G_n. Then we say that column *i* and column *j* are linked if and only if $\left|i-j\right| = 1$. Next, we should determine the

relative strength of these links. The way we defined the amplitudes for the columns suggests that we should define the strength of the link between column i and column j to be proportional to

$$\frac{e_{ij}}{\sqrt{N_i N_j}}, \tag{5.29}$$

where e_{ij} denotes the number of edges between column i and column j in the original graph. This analysis shows that all but one of the links between the adjacent columns of G_n are equivalent in strength. The one between column n and column n+1 is stronger than the others by a factor of $\sqrt{2}$. (This difference is sometimes called a defect.)

Then the quantum walk on G_n is reduced to the quantum walk on a "defective" line graph where each vertex c_i stands for column i of G_n and is represented by the state $|C_i\rangle$ in the column subspace. In particular, the vertex c_0 stands for *ENTRANCE* and the vertex c_{2n+1} stands for *EXIT*. The defective line graph for G_n is depicted in Figure 5.4.

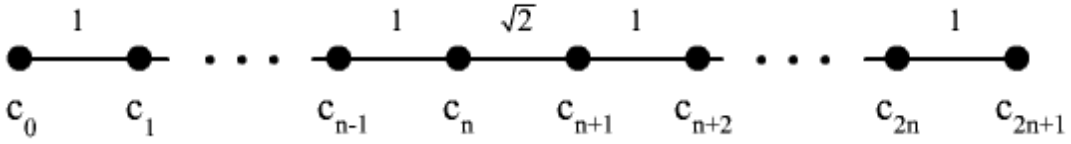

Figure 5.4. The defective line corresponding to the columns of G_n

5.4.2. Quantum Walk on the Defective Line

We have seen that the analysis of the quantum walk on G_n starting from *ENTRANCE* can be reduced to an analysis on a line with $2n+2$ vertices, one for each column of the original graph. This graph is called defective because of the edge between c_n and c_{n+1}, which has a transition probability larger than the others. In this section we will analyze continuous

time quantum walks on a defective line. While doing this, we will be referring to the graph G_{n-1}, instead of G_n, which will simplify the analysis. The defective line graph shown in Figure 5.5, is the reduced form of G_{n-1} and hence contains $2n$ vertices which we prefer to label as $c_1, c_2, .., c_n$. This time the vertex c_1 stands for *ENTRANCE* and the vertex c_{2n} stands for *EXIT*.

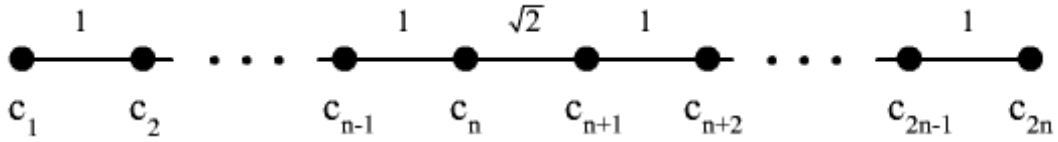

Figure 5.5. The defective line corresponding to the columns of G_{n-1}

The continuous time quantum random walk on this graph is governed by the Hamiltonian *H*, which is specified by the following non-zero terms and their symmetrics due to hermiticity. (We let $\gamma = 1$, for simplicity.)

$$H_{j,j+1} = \left\langle C_j \middle| H \middle| C_{j+1} \right\rangle = \begin{cases} 1 & 0 \le j \le 2n,\ j \ne n \\ \sqrt{2} & j = n \end{cases}, \tag{5.30}$$

An investigation of the eigenvectors of *H* will be useful in the following analysis of the quantum walk on the defective line. Introducing a reflection operator *R* such that $R\left|C_j\right\rangle = \left|C_{2n+1-j}\right\rangle$ will make this investigation easier. It can easily be verified that *R* commutes with *H* on the column subspace. The simultaneous diagonalization theorem states that commuting operators share the same eigenvectors [8]. Then the eigenvectors of *R* are also eigenvectors for *H*.

It is trivial that $R^2 = I$, so *R* has eigenvalues ± 1. The eigenvectors of *R* for the eigenvalue 1 should satisfy $R\left|S_1\right\rangle = \left|S_1\right\rangle$. These are of the form

$$\left|S_1\right\rangle = A_1\left|C_1\right\rangle + A_2\left|C_2\right\rangle + ... + A_n\left|C_n\right\rangle + A_n\left|C_{n+1}\right\rangle + A_{n-1}\left|C_{n+2}\right\rangle + ... + A_1\left|C_{2n}\right\rangle, \tag{5.31}$$

where $A_1,..,A_n$ are complex amplitudes. Similarly the equation $R\left|S_2\right\rangle = -\left|S_2\right\rangle$ should be satisfied by the eigenvectors of *R* for the eigenvalue -1. These are of the form

$$\left|S_2\right\rangle = A_1\left|C_1\right\rangle + A_2\left|C_2\right\rangle + ... + A_n\left|C_n\right\rangle - A_n\left|C_{n+1}\right\rangle - A_{n-1}\left|C_{n+2}\right\rangle - ... - A_1\left|C_{2n}\right\rangle. \tag{5.32}$$

The eigenvectors $\left|E\right\rangle = \sum_{k=1}^{2n} A_k\left|C_k\right\rangle$ of *H* should also be of the same form. Hence we have

$$\left\langle E\middle|C_{2n+1-k}\right\rangle = A_{2n+1-k} = \pm A_k. \tag{5.33}$$

If λ is the corresponding eigenvalue, then considering the structure of *H,* the equation $H\left|E\right\rangle = \lambda\left|E\right\rangle$ produces the following set of equations.

$$\lambda A_1 = A_2 \tag{5.34}$$

$$\lambda A_2 = A_1 + A_3 \tag{5.35}$$

$$\lambda A_3 = A_2 + A_4 \tag{5.36}$$

$$\vdots$$

$$\lambda A_{n-1} = A_{n-2} + A_n \tag{5.37}$$

$$\lambda A_n = A_{n-1} + \sqrt{2}A_{n+1} \tag{5.38}$$

The solution to the set of equations (5.33) – (5.38) is of the form,

$$\left\langle E\middle|C_k\right\rangle = A_k = \begin{cases} \sin(pk) & 1 \le k \le n \\ \pm\sin\left(p(2n+1-k)\right) & n+1 \le k \le 2n \end{cases} \tag{5.39}$$

where *p* is some real value and the eigenvalue λ corresponding to the eigenvector $\left|E\right\rangle$ is

$$\lambda = 2\cos(p). \tag{5.40}$$

The solution can be verified by the help of the following simple trigonometric identities.

$$\sin(x)+\sin(y)=2\sin\left(\frac{x+y}{2}\right)\cos\left(\frac{x-y}{2}\right) \tag{5.41}$$

$$\sin((k-1)p)+\sin((k+1)p)=2\sin(kp)\cos(p) \tag{5.42}$$

To investigate the exact value of p and hence the values for $E_1, E_2, .., E_{2n}$, we need to evaluate the quantization condition, which comes from the equation (5.38).

$$2\cos(p)\sin(np)=\sin((n-1)p)\pm\sqrt{2}\sin(np) \tag{5.43}$$

This can be simplified by use of (5.42) to

$$\sin((n-1)p)+\sin((n+1)p)=\sin((n-1)p)\pm\sqrt{2}\sin(np) \tag{5.44}$$

$$\sin((n+1)p)=\pm\sqrt{2}\sin(np) \tag{5.45}$$

So much about the eigenvectors of H is enough that we can continue to examine the quantum walk governed by H. In the following parts, we will be proving a set of theorems, which will ultimately lead to the desired result that the quantum walk on the defective line can reach the *EXIT* node (c_{2n}) in polynomial time, and so can the quantum walk on G_n.

Theorem 5.3. *Consider the quantum walk in* G_{n-1} *starting at the ENTRANCE. Let the walk run for a time t chosen uniformly in* $[0,\tau]$ *and then measure in the computational basis. If* $\tau \geq \frac{4n}{\varepsilon\,\Delta E}$ *for any constant* $\varepsilon > 0$, *where* ΔE *is the magnitude of the smallest gap between any pair of eigenvalues of H, the probability of finding the EXIT is greater than* $\frac{1}{2n}(1-\varepsilon)$.

Let us consider the walk on the defective line for G_{n-1}. Starting from the *ENTRANCE* node (c_1), letting the walk run for a time t results in a state equivalent to $e^{-iHt}|C_1\rangle$. At this final state the amplitude of the component parallel to *EXIT* node, $|C_{2n}\rangle$, is given by $\langle C_{2n}|e^{-iHt}|C_1\rangle$. Hence the probability of finding the *EXIT* when measured in the computational basis is $\left|\langle C_{2n}|e^{-iHt}|C_1\rangle\right|^2$.

If t is uniformly chosen in $[0,\tau]$, then the effective probability of finding the *EXIT* can be calculated by taking the average of $\left|\langle C_{2n}|e^{-iHt}|C_1\rangle\right|^2$ over the continuous interval $[0,\tau]$.

$$\begin{aligned} P(EXIT) = P(c_{2n}) &= \frac{1}{\tau}\int_0^\tau \left(\left|\langle C_{2n}|e^{-iHt}|C_1\rangle\right|^2\right)dt \\ &= \frac{1}{\tau}\int_0^\tau \left(\langle C_{2n}|e^{-iHt}|C_1\rangle\left(\langle C_{2n}|e^{-iHt}|C_1\rangle\right)^*\right)dt \end{aligned} \tag{5.46}$$

Let $\lambda_1, \lambda_2, ..$ denote the eigenvalues of H and let $|E_1\rangle, |E_2\rangle, ..$ denote the corresponding eigenvectors. If we apply spectral decomposition separately to the terms $\langle C_{2n}|e^{-iHt}|C_1\rangle$ and $\left(\langle C_{2n}|e^{-iHt}|C_1\rangle\right)^*$ then the expression becomes

$$\begin{aligned} &\frac{1}{\tau}\int_0^\tau \left(\left(\sum_i e^{-i\lambda_i t}\langle C_{2n}|\left(|E_i\rangle\langle E_i|\right)|C_1\rangle\right)\left(\sum_j e^{-i\lambda_j t}\langle C_{2n}|\left(|E_j\rangle\langle E_j|\right)|C_1\rangle\right)^*\right)dt \\ &= \frac{1}{\tau}\int_0^\tau \left(\left(\sum_i e^{-i\lambda_i t}\langle C_{2n}|E_i\rangle\langle E_i|C_1\rangle\right)\left(\sum_j e^{i\lambda_j t}\langle C_1|E_j\rangle\langle E_j|C_{2n}\rangle\right)\right)dt \\ &= \frac{1}{\tau}\int_0^\tau \left(\sum_{i,j} e^{-i(\lambda_i-\lambda_j)t}\langle C_{2n}|E_i\rangle\langle E_i|C_1\rangle\langle C_1|E_j\rangle\langle E_j|C_{2n}\rangle\right)dt \\ &= \frac{1}{\tau}\sum_{i,j}\int_0^\tau \left(e^{-i(\lambda_i-\lambda_j)t}\langle C_{2n}|E_i\rangle\langle E_i|C_1\rangle\langle C_1|E_j\rangle\langle E_j|C_{2n}\rangle\right)dt \end{aligned} \tag{5.47}$$

Now, the last summation can be broken into two parts; one for the case where $i = j$ and the other for the case where $i \neq j$. Rearranging the expression and taking the integrals in the appropriate way produces:

$$\sum_i \left|\langle E_i | C_1 \rangle\right|^2 \left|\langle E_i | C_{2n} \rangle\right|^2 + \sum_{i \neq j} \frac{\left(1 - e^{-i(\lambda_i - \lambda_j)\tau}\right)}{i(\lambda_i - \lambda_j)\tau} \langle C_{2n} | E_i \rangle \langle E_i | C_1 \rangle \langle C_1 | E_j \rangle \langle E_j | C_{2n} \rangle \tag{5.48}$$

By equation (5.33), we have $\langle E | C_1 \rangle = \pm \langle E | C_{2n} \rangle$ for any eigenvector $|E\rangle$ of *H*. Hence, the first term in (5.48) becomes

$$\sum_i \left|\langle E_i | C_1 \rangle\right|^2 \left|\langle E_i | C_{2n} \rangle\right|^2 = \sum_i \left|\langle E_i | C_1 \rangle\right|^4 . \tag{5.49}$$

The eigenvectors of a Hermitian matrix, when they are set as the columns of a matrix, are known to form a unitary operator by the spectral decomposition theorem. Hence, for any value of *i,* we have $\left(\sum_i \left|\langle E_i | C_1 \rangle\right|^2 \right) = 1$. Then it is easily established that

$$\sum_i \left|\langle E_i | C_1 \rangle\right|^4 \geq \frac{1}{2n} . \tag{5.50}$$

Recall that in Theorem 5.3, ΔE was defined to be the magnitude of the smallest gap between any pair of eigenvalues of the Hamiltonian *H*. Then using the facts $\left|1 - e^{-i(\lambda_i - \lambda_j)\tau}\right| \leq 2$ and $\left|i(\lambda_i - \lambda_j)\right| \geq \Delta E$, the second term in (5.48) can be bounded as follows.

$$\begin{aligned}
&\left|\sum_{i\neq j}\frac{\left(1-e^{-i(\lambda_i-\lambda_j)\tau}\right)}{i(\lambda_i-\lambda_j)\tau}\langle C_{2n}|E_i\rangle\langle E_i|C_1\rangle\langle C_1|E_j\rangle\langle E_j|C_{2n}\rangle\right| \\
&\leq \frac{2}{\tau\Delta E}\sum_{i,j}\left|\langle E_i|C_1\rangle\right|^2\left|\langle E_j|C_{2n}\rangle\right|^2 \\
&= \frac{2}{\tau\Delta E}\left(\sum_i\left|\langle E_i|C_1\rangle\right|^2\right)\left(\sum_j\left|\langle E_j|C_{2n}\rangle\right|^2\right) \\
&= \frac{2}{\tau\Delta E}
\end{aligned} \tag{5.51}$$

Finally, choosing $\tau \geq \frac{4n}{\varepsilon\Delta E}$ as Theorem 5.3 suggests, and using the results in (5.50) and (5.51), we have,

$$\frac{1}{\tau}\int_0^\tau dt\left|\langle C_{2n}|e^{-iHt}|C_1\rangle\right|^2 \geq \frac{1}{2n}-\frac{2}{\tau\Delta E}\geq\frac{1}{2n}(1-\varepsilon)\,. \tag{5.52}$$

Then the probability of finding the *EXIT* after running the walk for a time t chosen uniformly in $[0,\tau]$ where $\tau \geq \frac{4n}{\varepsilon\ \Delta E}$, is shown to be greater than $\frac{1}{2n}(1-\varepsilon)$. This result completes the proof for Theorem 5.3.

Note that if ΔE is too small then τ can become too large. In this case the above theorem can become practically useless. Therefore, we need to show that ΔE is at most polynomially small in terms of n and hence, τ is at most polynomially big. Theorem 5.4 states this.

Theorem 5.4. *The smallest gap between any pair of eigenvalues of the Hamiltonian H satisfies* $\Delta E > \frac{2\pi^2}{(1+\sqrt{2})n^3}+O\left(\frac{1}{n^4}\right)$.

We had shown that the eigenvalues of *H* are of the form $\lambda = 2\cos(p)$ where the values for *p* come from the roots of equation (5.45), which we refer to as the quantization condition. A rewriting of (5.45) gives

$$\frac{\sin((n+1)p)}{\sin(np)} = \pm\sqrt{2}\,. \tag{5.53}$$

In order to evaluate the spacings between the eigenvalues, we need to examine the roots of this equation. To give an idea about its behavior we sketch the left hand side of the equation for $n = 5$ in Figure 5.6 and for $n = 8$ in Figure 5.7. The two figures show similar characteristics on which we can make generalizations to build an analysis of the eigenvalues of the Hamiltonian *H*.

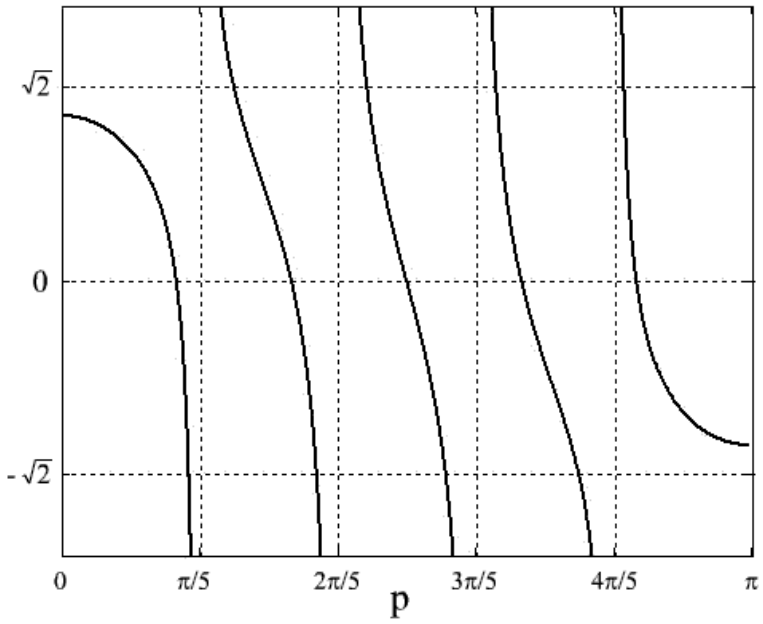

Figure 5.6. The sketch of $f(p) = \sin((n+1)p)/\sin(np)$ for $n = 5$

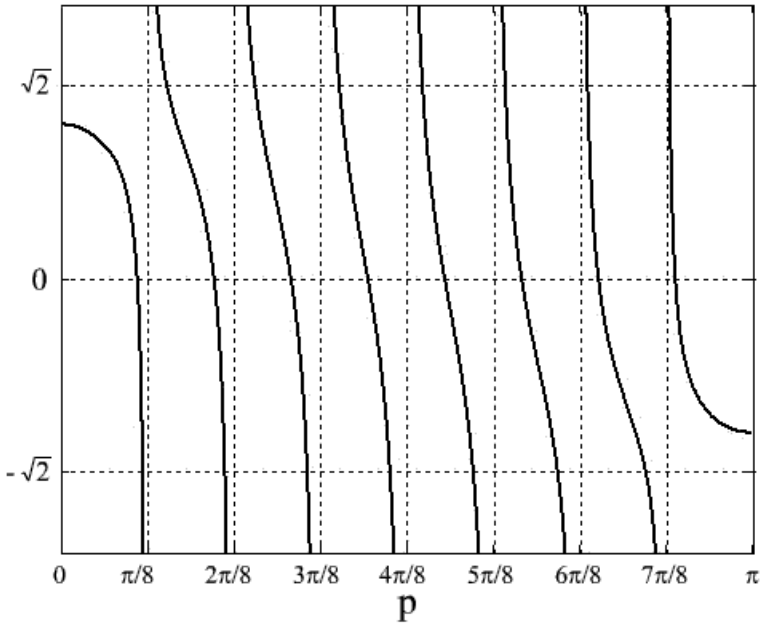

Figure 5.7. The sketch of $f(p)=\sin((n+1)p)/\sin(np)$ for $n=8$

It is seen in both graphs that the roots of the equation $f(p)=\sin((n+1)p)/\sin(np)=-\sqrt{2}$ lie to the left of the roots of the equation $\sin(np)=0$, which occur at $(\pi l/n)$, for $l=1,2,...,n-1$. Therefore we can say that $f(p)$ intersects $-\sqrt{2}$ for $p=(\pi l/n)-\delta$ where δ is non-negative. Replacing p with $(\pi l/n)-\delta$ and then rearranging the terms, Equation (5.53) with $-\sqrt{2}$ on the right hand side can now be rewritten as

$$-\sqrt{2}\sin n\delta=\sin\left(n\delta-\frac{l\pi}{n}+\delta\right). \tag{5.54}$$

Without loss of generality, we can assume $\delta=(c/n)+(d/n^2)+O(1/n^3)$. Then as $n\to\infty$ equation (5.47) becomes $-\sqrt{2}\sin(c)=\sin(c)$, which implies $c=0$. So we use $\delta=(d/n^2)+O(1/n^3)$ instead, to get

$$-\sqrt{2}\sin\left(\frac{d}{n}+O\left(\frac{1}{n^2}\right)\right)=\sin\left(\frac{d}{n}-\frac{l\pi}{n}+O\left(\frac{1}{n^2}\right)\right), \tag{5.55}$$

which as $n\to\infty$ gives $d=\dfrac{l\pi}{1+\sqrt{2}}$. Hence, we can state that the roots of (5.53) with $-\sqrt{2}$ on the right hand side are of the form

$$p=\frac{l\pi}{n}-\frac{l\pi}{(1+\sqrt{2})n^2}+O\left(\frac{1}{n^3}\right). \tag{5.56}$$

Let p' be a root of (5.53) with $-\sqrt{2}$ on the right hand side. Then p' is of the form shown in (5.56). Let p'' and p''' be the roots of (5.53) which lie respectively to the left and to the right of p'. It is clear in Figure 5.8 that the closest root to p' is either p'' or p''', both of which are roots of (5.53) with $+\sqrt{2}$ on the right hand side.

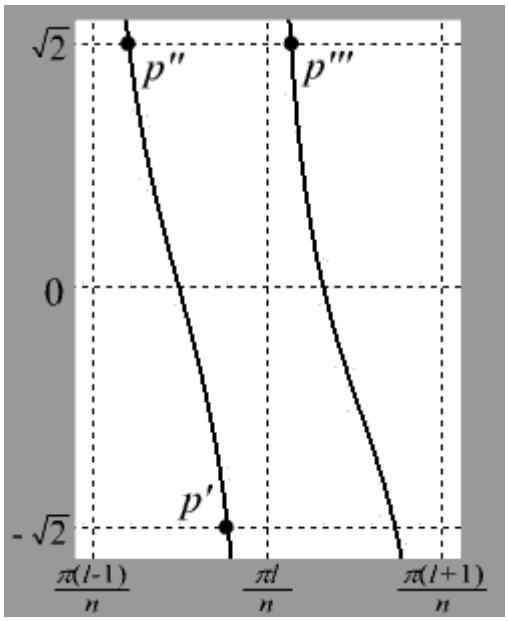

Figure 5.8. The roots p', p'' and p''' of the equation $\sin\big((n+1)p\big)/\sin(np)=\pm\sqrt{2}$

Note that p''' lies to the right of the zero of $\sin(np)$ at $p=(\pi l/n)$ and p'' lies to the left of the zero of $\sin((n+1)p)$ at $p=(\pi l/n+1)$. Therefore, we have $p''<\frac{l\pi}{n}-\frac{l\pi}{n^2}+O\left(\frac{1}{n^3}\right)$ and $p'''>\frac{l\pi}{n}$. From these, we can conclude the following for $l=1,2,...,n-1$:

$$p'-p''>\frac{l\pi\sqrt{2}}{(1+\sqrt{2})n^2}+O\left(\frac{1}{n^3}\right) \tag{5.57}$$

$$p'''-p'>\frac{l\pi}{(1+\sqrt{2})n^2}+O\left(\frac{1}{n^3}\right) \tag{5.58}$$

Thus, the smallest spacing between the roots of (5.53) is at least $\frac{\pi}{(1+\sqrt{2})n^2}+O\left(\frac{1}{n^3}\right)$ which is the value of $p'''-p'$ for $l=1$.

Now, we should translate the result about Δp, the minimum spacing between the roots of (5.46), to a result about the ΔE, the minimum spacing between the eigenvalues of *H*. It was shown that the eigenvalues of *H* are of the form $2\cos(p)$, where p is a root of (5.53). Then ΔE and Δp are related by $\Delta E=2\cos(p+\Delta p)-2\cos(p)$. For small values of Δp, this relation becomes

$$\Delta E=2\left|\Delta p\sin(p)\right|+O\left((\Delta p)^2\right). \tag{5.59}$$

The factor $\sin(p)=\sin\left(\pi l/n+O\left(1/n^2\right)\right)$ is smallest when $l=1$. If we place the smallest possible terms for Δp and $\sin(p)$ in equation (5.59) we can find a lower bound for the value of ΔE. Then, for sufficiently large *n* we write

$$\Delta E>\frac{2\pi^2}{(1+\sqrt{2})n^3}+O\left(\frac{1}{n^4}\right)>\frac{8}{n^3}. \tag{5.60}$$

This concludes the proof of Theorem 5.4, which stated that the smallest gap between any pair of eigenvalues of H is only polynomially small. Now we have sufficient results to form a general statement about the quantum walks on the glued trees.

Theorem 5.5. *For sufficiently large n, running the quantum walk on* G_n *for a time chosen uniformly in* $\left[0, \frac{n^4}{2\varepsilon}\right]$ *and then measuring in the computational basis, yields a probability of finding the EXIT that is greater than* $\frac{1}{2n}(1-\varepsilon)$.

Theorem 5.4. states that the spacing ΔE between the eigenvalues of H is only polynomially small. With this insight, we can state that $\tau \geq \frac{4n}{\varepsilon\ \Delta E}$ is only polynomially large. Hence, by the statement of Theorem 5.3, for any graph of the form G_n, we can efficiently run the quantum walk, with use of the methods introduced in Section 5.3, for a randomly decided, polynomially big time. Measuring in the computational basis, then gives a probability of finding the *EXIT*, which is greater than $\frac{1}{2n}(1-\varepsilon)$. We can check whether the observed state is *EXIT*. A constant number of black box queries may be used to see if the degree of the observed state is equal to two, in which case we can safely conclude that the observed state is *EXIT* as long as it is not equal to *ENTRANCE*. (Note that only these two nodes have degrees equal to two in a graph of the form G_n.)

A success probability which is arbitrarily close to 1 can be achieved by the repetition of this process by a polynomially big number of times. To conclude, the quantum random walk algorithm can solve the black box graph traversal problem for the graphs of the form G_n to an arbitrary degree of certainty, with use of polynomially big number of black box queries and other one and two qubit gates.

5.5. The Classical Lower Bound

In Section 5.4 we have derived a result that a quantum algorithm depending on continuous time quantum random walks can efficiently solve the graph traversal problem on graphs of the form G_n to an arbitrary degree of certainty. (In [50], it is stated that a discrete time quantum walk employing the Grover coin can serve for the same task as well.) The importance of this result is due to the fact that no classical algorithm can do the same in sub-exponential time. In this section, we will show the truth of this statement. In particular, we will be constructing a proof for the following theorem.

Theorem 5.6. *Any classical algorithm that makes at most* $2^{n/6}$ *queries to the oracle finds the EXIT with probability at most* $4 \cdot 2^{-n/6}$.

In [3], a set of games are introduced which serves for the proof of Theorem 5.6. We will follow the same way and go through these games, the first of which will be equivalent to the graph traversal problem. Each new game will be essentially as easy to win as the previous one. At the end, we will try to show that the easiest game cannot be won in subexponential time, which is the desired result.

In the following, we will be using $\mathrm{P}_X^G(A) = \Pr_{names}\left[A \text{ wins game } X \text{ on graph } G\right]$ to denote the probability that the algorithm *A* wins the game *X* played on the graph *G* where the vertices are randomly named.

Let us begin with the first game.

5.5.1. Game 1: Find the Exit

The following game is equivalent to the graph traversal problem.

Given a randomly chosen graph G_n such that each vertex has a distinct 2*n*-bit string as its name and the *ENTRANCE* vertex has the name 00..0. (The string 11..1 is not used as the name of any vertex.) Also given a black box for this graph, which returns the names of the neighbors of a vertex if it is queried with the name of that vertex. An algorithm wins the game if it ever sends the black box the name of the *EXIT* vertex.

In this game, there is no restriction on the inputs to the black box, so an algorithm could traverse a disconnected subgraph of G_n. But since the number of strings that can be written in 2*n* bits is exponentially larger than the number of vertices in the graph, it is highly unlikely that any algorithm could ever guess the name of a vertex which was not sent by the oracle. Therefore Game 1 is essentially equivalent to the following game.

5.5.2. Game 2: Find a Path to the Exit

Game 2 is defined in the same way as Game 1 with the exception that a string to be sent to the black box should either be the name of the *ENTRANCE* node or the name of a vertex that has previously been returned by the black box.

Let A be an algorithm for Game 1. One can define an algorithm A' for Game 2 in the same way as A, but in the cases that A sends the black box a string which is not the name of the *ENTRANCE* node or a name previously returned by the oracle, A' considers the result of the query as 11..1.

The two algorithms A and A' will have similar behavior unless A guesses the name of a new vertex correctly. The probability, $\mathrm{P}_t(guess)$, that A does so at least once in *t* trials is given by

$$\mathrm{P}_t(guess) = 1 - \left(\frac{N_{strings} - N_{vertices}}{N_{strings}} \right)^t = 1 - \left(\frac{\left(2^{2n} - 1\right) - \left(2^{n+2} - 2\right)}{(2^{2n} - 1)} \right)^t \cong \frac{t\left(2^{n+2} - 2\right)}{2^{2n} - 1}. \qquad (5.61)$$

which is in the order of $O\left(t/2^n\right)$. ($N_{strings}$ is the number of strings different from 11...1 that can be written in $2n$ bits and $N_{vertices}$ is the number of vertices in the graph G_n.) Therefore the success probabilities $\mathrm{P}_1^G(A)$ and $\mathrm{P}_2^G(A')$ differ by a factor at this order.

$$\mathrm{P}_1^G(A) \le \mathrm{P}_2^G(A) + O\left(\frac{t}{2^n}\right) \tag{5.62}$$

It should be noted that winning Game 2 is essentially as easy as solving the graph traversal problem by querying the black box with randomly chosen strings at each step. However if an algorithm can develop a mechanism of inference about its position of the known vertices on the graph then it can perform better than random by querying the black box for carefully chosen vertices. An algorithm knows the column on which a particular vertex *v* is if it has seen a path shorter than $n+2$ edges from *ENTRANCE* to *v*. This is of no use for the columns which lie to the right of column $n+1$. However if an algorithm ever sees a cycle in the graph then the structure of this cycle may reveal some non-trivial information about the position of the vertices on the graph. Of course this information may not be so critical in winning Game 2, however we should take this probability into the account to find an upper bound on the probability of solving the graph traversal. The definition of the next game is a step in this way.

5.5.3. Game 3: Exit or Cycle

Game 3 is defined in the same way as Game 2 except the winning condition. An algorithm wins Game 3 if it ever sends the oracle the name of the exit vertex, or if the subgraph it has seen contains a cycle.

Trivially, Game 3 is easier to win than Game 2 is as stated in (5.63). Therefore an upper bound on the probability of winning this game in *t* steps also bounds the probability that one can solve the graph traversal problem with use of *t* queries.

$$\mathrm{P}_2^G(A) \le \mathrm{P}_3^G(A) \tag{5.63}$$

We define an *embedding* of a rooted binary tree T into a graph G to be a mapping π, from the vertices of T to the vertices of G such that $\pi(ROOT) = ENTRANCE$ and for all vertices u and v that are neighbors in T, $\pi(u)$ and $\pi(v)$ are neighbors in G. We say that an embedding of T is *proper* if $\pi(u) \neq \pi(v)$ for $u \neq v$. A tree T is said to exit a graph G under an embedding π if $\pi(v) = EXIT$ for some $v \in T$. It is not hard to show that the subgraph an algorithm sees must be a proper random embedding of a rooted binary tree if it could not win Game 3. The definition of the next game depends on this observation.

5.5.4. Game 4: Exit or Cycle With a Binary Tree

In this game, an algorithm simply produces a rooted binary tree T with t vertices. Every vertex of T which is not a leaf is required to have two children. An embedding π of T into G_n is produced randomly. The algorithm wins Game 4 if π is an improper embedding or if T exits G_n under π.

A random embedding of a tree T is obtained by setting $\pi(ROOT)$ = *ENTRANCE* and then mapping the rest of T into G at random. For a binary tree in particular, the following algorithm can be used to obtain a random embedding.

1. Label the root of T as 0, and label other vertices of T with consecutive integers so that if vertex i lies on the path from the root to the vertex j, then $i < j$.
2. Set $\pi(0) = ENTRANCE$.
3. Let i and j be the neighbors of 0 in T.
4. Let u and v be the neighbors of *ENTRANCE* in G.
5. Set $\pi(i) = u, \pi(j) = v$ or $\pi(i) = v, \pi(j) = u$, with probability $1/2$ for each.
6. For $i = 1,2,3..$ if vertex i is not a leaf in T and $\pi(i)$ is not ENTRANCE or EXIT,

(a) Let j and k denote the children, and l denote the parent of vertex i.

(b) Let u and v be the neighbors of $\pi(i)$ in G other than $\pi(l)$.

(c) Set $\pi(i)=u, \pi(j)=v$ or $\pi(i)=v, \pi(j)=u$, with probability $1/2$ for each.

Let A be an algorithm which wins Game 3 that uses at most t queries to the black box. One can define an algorithm A' for Game 4 to generate a random tree by simulating A as follows. Suppose that a vertex a in graph G corresponds to vertex a' in the tree that A' is generating. If A asks the black box for the names of the neighbors of a, A' generates two unused names b' and c' at random and uses them as neighbors of a'. Now corresponding to b and c, the neighbors of a in G, the tree has b' and c'. It is easy to verify that using the tree generated by A' wins Game 4 if and only if A wins Game 3. Therefore we say $\mathrm{P}_3^G(A) = \mathrm{P}_4^G(A')$, which means that Game 3 and Game 4 are equivalent.

Now, we have a chain of games, the first of which is equivalent to the graph traversal problem. Assuming n is large Game 1 and Game 2 are essentially equal in difficulty. The easiest ones to win are Game 3 and Game 4. Therefore an upper bound on the probability of winning Game 4 in a finite number of steps is also an upper bound on the probability that a classical algorithm can solve the graph traversal problem in that many steps. Therefore we are going to establish such a bound. In particular we will be proving that the expected probability that a rooted tree T of at most $2^{n/6}$ vertices can win Game 4 on a randomly selected graph G_n is at most $3 \cdot 2^{n/6}$.

Let T be a tree with t vertices ($t \le 2^{n/6}$). T wins Game 4 if it satisfies one of the two winning conditions. Let π be a random embedding of T into G_n. Then $\pi(T)$ is the image of T in G_n under π. T wins Game 4 if $\pi(T)$ contains the *EXIT* node of G_n, which is the first winning condition. Note that in order for $\pi(T)$ to contain a node in the k'th column of G_n ($n+1 < k \le 2n+2$), there should be a subbranch in T, whose image under π is a path from column $n+1$ to column k of G_n. The probability of this for a single branch of length

$k-(n+1)$ is $1/2^{k-(n+1)}$, since the random embedding of this branch should choose to move right $k-(n+1)$ times in a row to reach the k'th column. Since there are at most t tries on each branch of T and there are at most t such branches, the probability that $\pi(T)$ contains a node in the k'th column is bounded by $t^2/2^{k-(n+1)}$. In particular, a node in the $3n/2$'th column of G_n can occur in $\pi(T)$ with a probability at most $t^2 2^{1-n/2}$, and we can take this as an upper bound on the probability that T satisfies the first condition.

T wins Game 4 also if $\pi(T)$ contains a cycle, which is the second winning condition. If $\pi(T)$ contains a cycle, then there are two vertices a,b in T such that $\pi(a)=\pi(b)$. If P is the path between a and b in T, then $\pi(P)$ is a cycle in G_n. Let c be the vertex nearest to the root in $\pi(P)$ and divide $\pi(P)$ into two paths $\pi(P_1)$, from c to a and $\pi(P_2)$, from c to b. Without loss of generality one can take $\pi(c)$ to be in the left half. Consider Figure 5.9 where every triangle stands for a subtree of depth $n/2$.

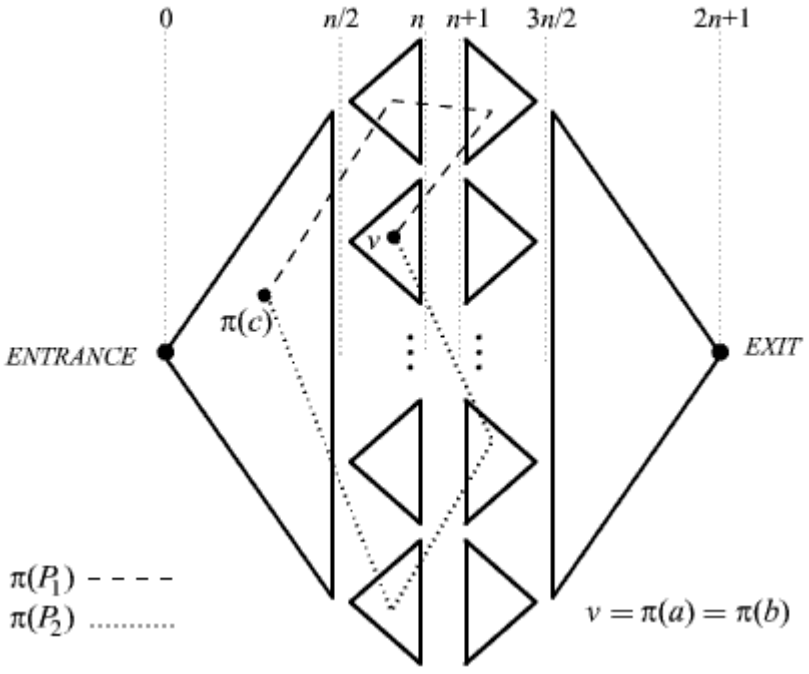

Figure 5.9. The cycle depicted on the subtrees of G_n

The $2^{n/2}$ subtrees populating the columns $n+1$ through $3n/2$ are named $S_1, S_2, .., S_{2^{n/2}}$ and the $2^{n/2}$ subtrees populating the columns $n/2$ through n are named $S_1', S_2', .., S_{2^{n/2}}'$ Both $\pi(P_1)$ and $\pi(P_2)$ visit a sequence of these subtrees and since $\pi(a) = \pi(b)$, these sequences should end with the same subtree. The other possibility where $\pi(a) = \pi(b)$ lies in a column greater than $3n/2$ or less than $n/2$ is bounded by the same probability as the one calculated for a path from the column $n+1$ to the column $3n/2$.

If both sequences had contained only one subtree, then all the vertices visited should be in the left half, which would not create a loop. So at least one of the sequences contains more than one subtree. The probability that the last terms in the two sequences are the same is bounded by $2^{n/2}/(2^n - t)$, because of the random cycle that connects the two halves of G_n. As long as $t \le 2^{n-1}$, it is guaranteed that $2^{n/2}/(2^n - t) \le 2 \cdot 2^{-n/2}$. Since the number of paths is $\binom{t}{2}$, which is less than t^2, one can conclude that the probability of a cycle is less than $t^2 \cdot 2^{-n/2}$.

Then the probability that a rooted tree *T* with *t* vertices can win Game 4 on a randomly selected graph G_n is bounded by $t^2 \cdot 2^{1-n/2} + t^2 \cdot 2^{-n/2}$. If a tree with at most $2^{n/6}$ vertices is considered, $3 \cdot 2^{n/6}$ can be set as an upper bound on that probability. This completes the proof of Theorem 5.6. It is shown that a classical algorithm for solving the graph traversal problem requires exponential time.

6. AN IMPROVEMENT ON QUANTUM WALK SIMULATIONS

In the preceding chapter we analyzed the results demonstrated in [3], where a quantum algorithm depending on a black box based implementation of a continuous time quantum walk was shown to provide an exponential gain against the best classical algorithm for a graph traversal problem. In this chapter, we present an optimized version of the quantum random walk implementation used in [3]. Our approach can be generalized to optimize any quantum simulation in which the linear combination rule is used to simulate a collection of constituent Hamiltonians. The method involves manipulation of the order in which the constituent Hamiltonians are simulated for small durations in the iterative step of the simulation algorithm. An analysis to illustrate the benefits of the new approach in oracle-based simulations is also given.

In Section 6.1 we will construct a circuit model for the quantum walk implementation introduced in Section 5.3. Then a method for optimizing this implementation will be demonstrated in Section 6.2. An analysis of the gain of this technique in various black box scenarios will be presented in Section 6.3 and Section 6.4 will conclude the discussion.

6.1. The Circuit Model for Quantum Walk Simulation

Circuit models are often useful to visualize quantum algorithms. In this section we will construct a circuit model for the quantum walk implementation of Section 5.3. Let us first recall the details of the proposed implementation.

The quantum walk on a graph G was shown to be governed by a Hamiltonian H which could be written as a sum of smaller terms. We have

$$H = \sum_{c \in L} H_c \,, \tag{6.1}$$

where L is a set of k labels used for the edges of G. For each $c \in L$, the corresponding local term, H_c is of the form

$$H_c = V_c^\dagger T V_c, \tag{6.2}$$

where for each $c \in L$, V_c is the component that queries the black box that satisfies

$$V_c^\dagger = V_c. \tag{6.3}$$

and T is the operator given by $T = S^{(1,\,m+1)} \otimes S^{(2,\,m+2)} \otimes \ldots \otimes S^{(m,\,2m)} \otimes |0\rangle\langle 0|$ where $S^{(u,v.)}$ denotes the swap operator acting on the bits u and v. Recall also that the circuit in Figure 6.1 simulates T for an arbitrary duration Δt on the states of the form $|a,b,r\rangle$. The last qubit initially set to $|0\rangle$ is an ancilla bit and it is uncomputed at the end.

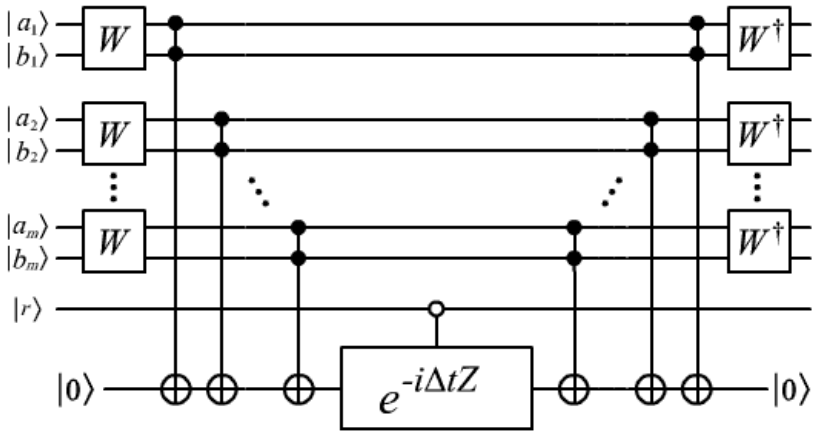

Figure 6.1. The circuit for simulating $T = S^{(1,\,m+1)} \otimes S^{(2,\,m+2)} \otimes \ldots \otimes S^{(m,\,2m)} \otimes |0\rangle\langle 0|$

The unitary conjugation rule suggests that

$$e^{-i\left(V_c^\dagger T V_c\right)\Delta t} = V_c^\dagger e^{-iT\Delta t} V_c. \tag{6.4}$$

Then, if the circuit for simulating T for a duration Δt is represented by a component labeled as $e^{-iT\Delta t}$, we can construct the circuit which simulates $H_c = V_c^{\dagger} T V_c$ as in Figure 6.2.

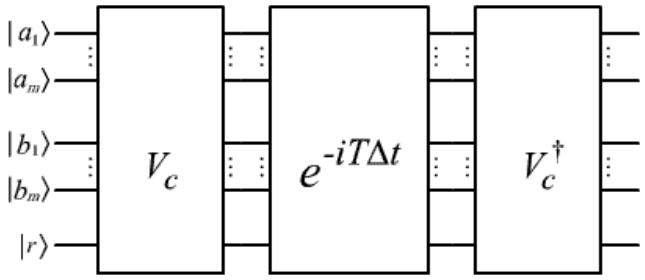

Figure 6.2. The circuit simulating $H_c = V_c^{\dagger} T V_c$ for duration Δt

Let $L = \{l_1, l_2, .., l_k\}$, in which case it is natural to name the local Hamiltonians as $H_1, H_2, .., H_k$, so that $H = H_1 + H_2 + .. + H_k$. Then by linear combination, we have

$$\left\| e^{-i(H_1 + ... + H_k)t} - \left(e^{-iH_1 t/r} \cdots e^{-iH_k t/r} \right)^r \right\| = O\left(\frac{k^2 l t^2}{r} \right). \tag{6.5}$$

If only non-commuting pairs of constituent Hamiltonians are considered, the right hand side of (6.5) becomes $O\left(klt^2/r\right)$. Then, depending on the desired accuracy, the total simulation time t is divided into r slices and simulated in r iterations of $\left(e^{-iH_1 t/r} e^{-iH_2 t/r} \cdots e^{-iH_k t/r} \right)$. All of these iterations are identical and the overall simulation can be represented as in Figure 6.3, where we take $\Delta t = t/r$ and let the components $e^{-iH_c \Delta t}$, denote the circuit in Figure 6.2 for $l_c \in L$.

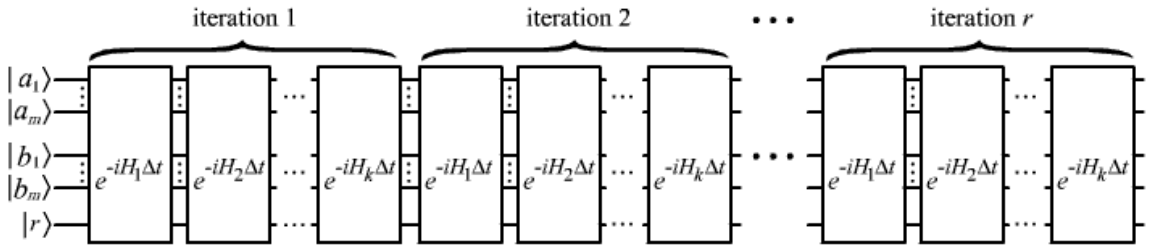

Figure 6.3. The simulation of quantum walk in r iterations of $\left(e^{-iH_1 t/r} e^{-iH_2 t/r} \cdots e^{-iH_k t/r} \right)$

In Sections 5.3 and 5.4, it was shown that running the circuit in Figure 6.3 requires $O\left(k^2t^2/\varepsilon\right)$ black box queries and $O\left(k^2t^2m/\varepsilon\right)$ auxiliary operations for symmetric labeling. It was also shown that for the graphs of the form G_n where n is sufficiently large, running the circuit designed for a time t uniformly chosen in $\left[0,n^4/2\varepsilon\right]$ on the initial state $\left|a,b,r\right\rangle = \left|ENTRANCE,0,0\right\rangle$, and then measuring in the computational basis yields a probability of finding the *EXIT* that is greater than $\frac{1}{2n}(1-2\varepsilon)$. Therefore, the graph traversal problem on the graphs G_n (which is classically hard) can be solved to arbitrary accuracy by a polynomially big number of repetitions of this process.

6.2. An Improvement on the Quantum Walk Implementation

The complexity of a quantum algorithm is measured in terms of the number of conventional quantum gates and oracle calls (if any) that it needs to solve a problem. A reduction in these numbers is often regarded as a valuable optimization. The quantum walk simulation algorithm introduced above is open to such optimization in several ways. Here is a discussion of such an effort to improve this algorithm.

The simulation algorithm is based on the trivial fact that the Hamiltonian H, which governs the random walk, can be written as a sum of several smaller Hamiltonians H_c where $c \in L$. The iterative part of the algorithm is the successive simulation of these smaller Hamiltonians for small time slices. The order in which these simulations will take place is a natural candidate to serve as an instrument for a potential optimization. In the following we will first show that manipulating this order does not effect the expected accuracy of the overall simulation algorithm and next we will build an optimization method depending on this observation.

The terms $e^{-iH_1\Delta t}, e^{-iH_2\Delta t}, .., e^{-iH_q\Delta t}$ can be ordered in $q!$ different ways. Let each of $a_1, a_2, .., a_r$ denote the product of these terms in arbitrary orders. ($a_1, a_2, .., a_r$ can be distinct or not.) By (3.13) for each $1 \le i \le r$, we have

$$e^{-i\left(H_1+H_2+...+H_q\right)\Delta t} = a_i + b \tag{6.6}$$

where b denotes the error term $O\left(q^2 \ell \left(\frac{t}{r}\right)^2\right)$. Now we can write

$$\begin{aligned} \left(e^{-i\left(H_1+H_2+...+H_q\right)\Delta t}\right)^r &= \prod_{i=1}^{r}\left(a_i + b\right) \\ &= \prod_{i=1}^{r} a_i + \sum_{i=1}^{r}\left(\prod_{\substack{1\le j\le r \\ j\ne i}} a_j\right) b + ... \\ &= \prod_{i=1}^{r} a_i + O(rb) \\ &= \prod_{i=1}^{r} a_i + O\left(\frac{q^2 \ell t^2}{r}\right) \end{aligned} \tag{6.7}$$

The error term is in the same order as in the original case. (In particular case of simulating random walk on a graph G_n, this error term reduces to $O\left(q\ell t^2 / r\right)$ since only $O(q)$ pairs of q constituents fail to commute.)

$$\left\| \left(e^{-i\left(H_1+H_2+...+H_q\right)\Delta t}\right)^r - \prod_{k=1}^{r} a_k \right\| = O\left(\frac{q^2 \ell t^2}{r}\right) \tag{6.8}$$

Therefore, we can state that alternating the orders at each iteration does not affect the average accuracy of the simulation algorithm. Let us now see how this observation can be used to build an optimization method on the simulation algorithm.

A simple idea is to start each iteration (with the trivial exception of the first one) with the simulation of that constituent H_c which was the last in the previous iteration. This can be achieved by simply reversing the order at each iteration as in (6.9).

$$a_i = \begin{cases} \left(e^{-iH_1\Delta t}e^{-iH_1\Delta t}..e^{-iH_q\Delta t}\right) & \text{if } i \text{ is odd} \\ \left(e^{-iH_q\Delta t}e^{-iH_{q-1}\Delta t}..e^{-iH_1\Delta t}\right) & \text{if } i \text{ is even} \end{cases} \tag{6.9}$$

Then the overall algorithm runs by simulating the constituent Hamiltonians in the order specified by the string $H_1, H_2, .., H_k, H_k, H_{k-1}, .., H_1, H_1, H_2, .., H_k, ..$. If this method is applied, two successive simulations of the same Hamiltonian occurs $r-1$ times, like in the cases of $.., H_k, H_k, ..$ and $.., H_1, H_1, ..$. Such successive simulations of a Hamiltonian, which is of the form $H_c = V_c^\dagger T V_c$, bring in a potential for optimization due to the fact that $V_c V_c^\dagger = I$.

$$H_c H_c = V_c^\dagger T V_c V_c^\dagger T V_c = V_c^\dagger T T V_c . \tag{6.10}$$

With the successive simulations of $H_c = V_c^\dagger T V_c$, one can save two black box queries by simply canceling $V_c V_c^\dagger = I$. Therefore, the circuit that simulates the Hamiltonian $H_c H_c$ can be constructed as in Figure 6.4.

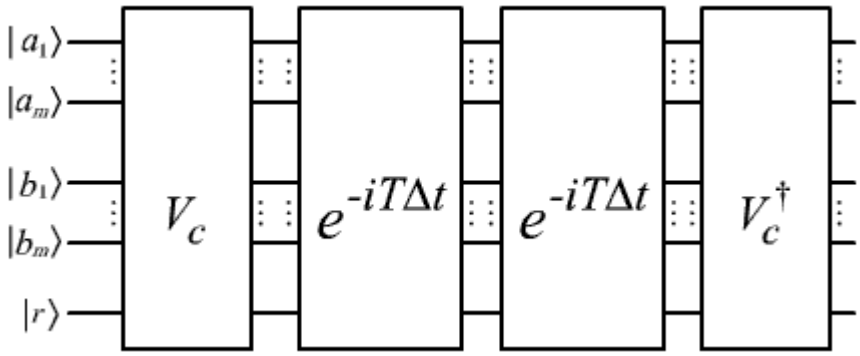

Figure 6.4. Two successive simulations of the Hamiltonian $H_c = V_c^\dagger T V_c$

With this optimization, the number of oracle calls needed for these successive simulations is reduced by half. The ratio of the improvement in the overall algorithm is dependent on k, the number of local interactions that make up the global Hamiltonian H. Each of the r iterations of the old method requires $2k$ oracle calls, while this number is reduced to $2k-1$ for the first and last of the r iterations, and to $2k-2$ for the others with the arrangement in the order of simulations. The maximum vertex degree for the sort of graphs contained in our problem is three, so k can be taken to be 3 for this problem. In this case, the total number of oracle calls is reduced by a factor in the order of one thirds of that number for the original algorithm, which is a valuable optimization, considering how hard it is to build quantum circuits. However the circuit shown in Fig. 6.4 is open to further improvement. A more detailed look (as in Figure 6.5) at the center of this circuit would reveal the idea.

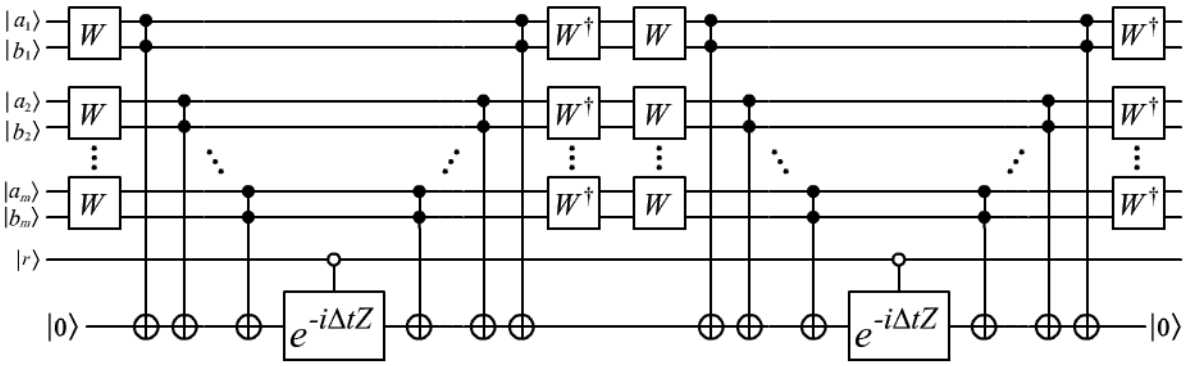

Figure 6.5. The circuit representation for two successive simulations of T

Figure 6.5 offers an opportunity to improve the circuit by canceling the unitary operator $W^{\otimes m}$ and its adjoint, which, when applied one after another, act as the identity operator. Then a further improvement becomes trivial by canceling the Toffoli gates to produce a circuit where two of the components for simulating the Pauli Z operator for duration Δt come together at the center. A natural choice is to replace these gates with a simulation of the Pauli Z operator for duration $2\Delta t$. Then we end up with a circuit like the one in Figure 6.6.

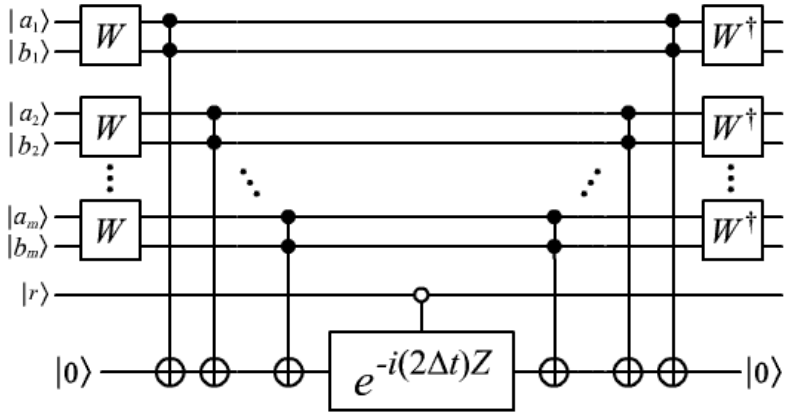

Figure 6.6. The circuit that simulates T for duration $2\Delta t$

Now it is also apparent that the circuit in Figure 6.4 simulates H_c for time $2\Delta t$. With one more arrangement we replace the successive terms, $..,H_c,H_c,..$ in the sequence $H_1,H_2,..,H_k,H_k,H_{k-1},..,H_1,H_1,H_2,..,H_k,..$ with H_c^2, which stands for the simulation of the Hamiltonian H_c for a duration twice as large as that for others. Then we get the sequence $H_1,H_2,..,H_k^2,H_{k-1},..,H_1^2,H_2,..,H_k^2,..$. Note that the last term of such a sequence is determined by the parity of r. The circuit for implementing the quantum walk with the new method is depicted in Figure 6.7 for the case where $k=3$ and r is even. Note that in this case the iterations are not separated by strict lines.

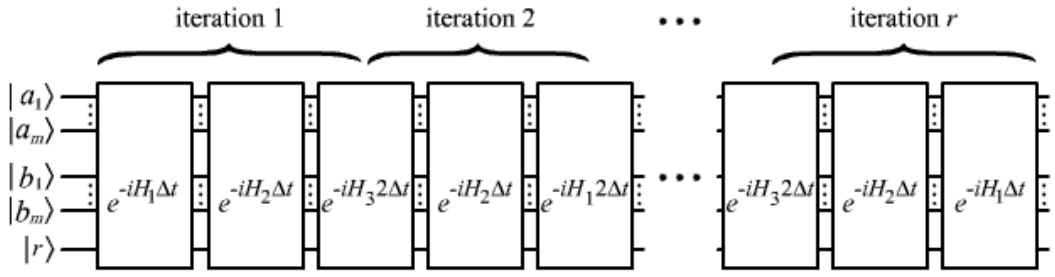

Figure 6.7. The circuit for simulating the sequence $H_1,H_2,..,H_k^2,H_{k-1},..,H_1^2,H_2,..,H_k^2,..$

This method brings an improvement over the old one, not only because it is computationally less expensive, but also because it eliminates more failures due to potential problems in physical implementations.

6.3. Performance Analyses for Various Black Box Settings

The method introduced can be more beneficial for implementing the quantum walk with a slightly modified version of the original black box settings. This section briefly discusses several such cases.

As the first example, consider a scenario that is the same as the original one, except that this time, the costs for the black box queries are not uniform. That is, different costs are associated with querying the black box for different edge labels $c \in L$. In this case, the improved algorithm performs better if it economizes on the types of calls which are more expensive. Suppose that the two most expensive types of queries are those associated to the labels l_a and l_b in decreasing order of costs. To achieve the best performance, the simulation should start the first of the *r* iterations with the second most expensive label l_b and end with the most expensive one l_a, not caring about the order of other labels between the two. Then reversing this order at each iteration would produce a sequence like $H_b,..,H_a^2,..,H_b^2,..,H_a^2,..$ thus at each iteration it saves two queries which are of one of the two most expensive types l_a and l_b.

In this scenario, the ratio of improvement with respect to the original method is dependent on how the costs are distributed. A simplified scenario contains three labels l_a, l_b and l_c, two of which (l_a and l_b) have the same cost C_a, which is *z* times as big as the cost C_c of the third one (l_c). The original method would use two queries for each label at each iteration, and thus would have a total query cost of $4rC_a + 2rC_c$. With the method we propose, $r-1$ successive simulations are transformed to longer simulations, hence saving two queries for a label, which we arrange to be one of the expensive ones. Therefore the total saving from the query cost is $2(r-1)C_a$. Then the total query cost of the proposed method is $(4rC_a + 2rC_c) - 2(r-1)C_a = (2r+2)C_a + 2rC_c$.

The ratio R of the cost for the proposed method to the cost for the original one varies according to the number of iterations *r*, and the ratio of the costs, $z = C_a / C_c$. A sketch of R against $z = C_a / C_c$ is given in Figure 6.8, where *r* is assumed to be large. It is seen that for uniform label costs, the improvement over the original algorithm is in the order of one thirds of the total cost for the original algorithm, while that ratio increases to a half with the increase in the ratio of the non-uniform costs.

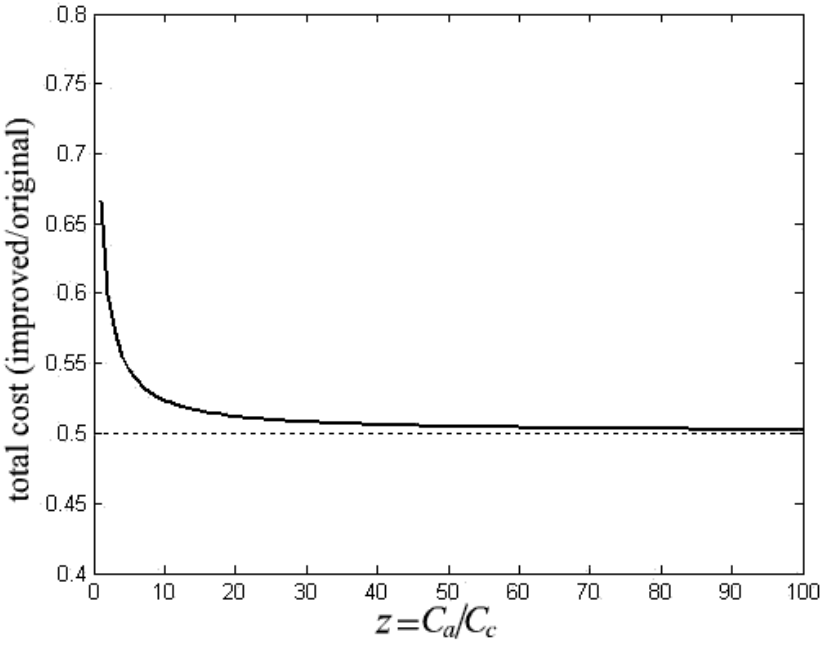

Figure 6.8. The ratio of total costs sketched against the ratio of query costs

Another interesting case occurs when the number of black box queries that a quantum walk implementation can perform is limited. Several variations of this scenario can be considered. The most interesting alternative imposes independent limits for queries of different labels. The performance criterion in such a case is the precision of the simulation, which is a polynomial function of the number of iterations that the simulation can run. The maximum number of iterations that the original algorithm can run is half the number of the allowed queries for the label with the tightest limitation, since with this method each iteration would need two queries of each type. The method introduced in Section 6.2 can be adjusted to

run for twice as many iterations (if limitations on other labels are not so tight) and hence returns more precise results. For a better comparison of the two algorithms, the maximum number of queries allowed for each label should be taken to be the same. The number of iterations that the original algorithm can run is half that number. The performance of the improved algorithm is dependent on the number of labels k, as depicted in Figure 6.9. For large values of k, the two algorithms return similar results, while for small values, the method we propose can run notably more iterations, which means better precision.

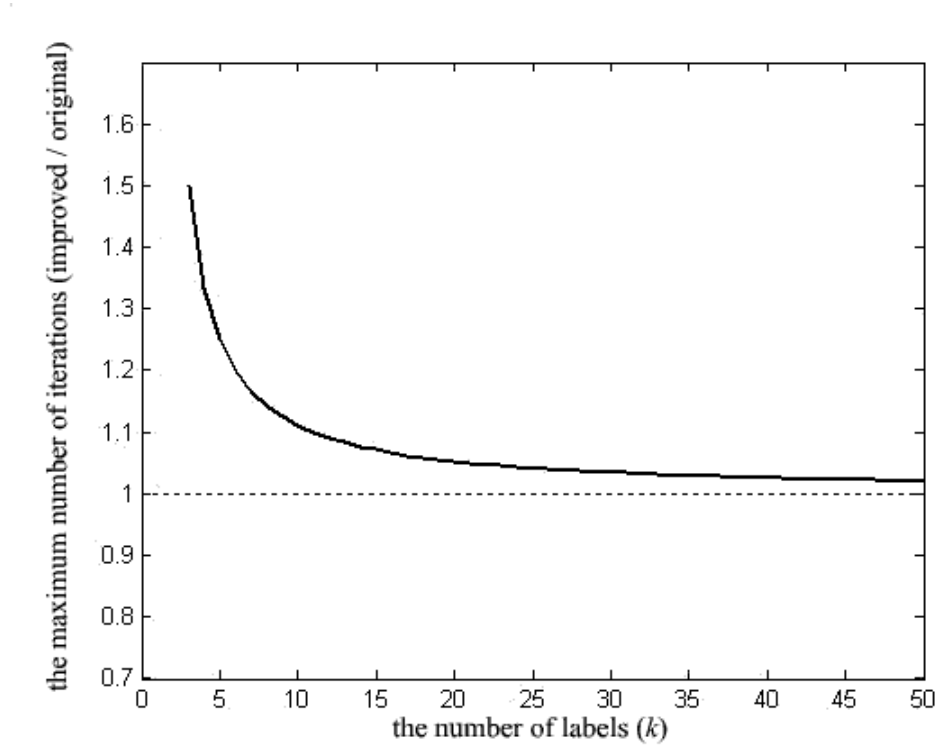

Figure 6.9. The ratio of maximum number of iterations sketched against k

Various other scenarios can be constructed, where the improved algorithm can be shown to perform significantly better than the original one. The overall improvement in complexity, together with a flexibility that can be used in favor of more preferable types of queries, can be seen to be the reasons for that improved performance.

7. CONCLUSION

In this work, we aimed to develop an understanding of the techniques used in demonstrating exponential algorithmic speedup by quantum walks. For this purpose, we examined the necessary simulation techniques and some of the many ideas in the domain of quantum random walks. Then we presented a detailed analysis of how to achieve the exponential algorithmic speedup by quantum walks. We also presented an optimization technique to reduce the computational cost of the quantum walk implementation used in this process.

It is evident that the optimization technique we presented is not limited to the simulation of quantum random walks, but it can be generalized to optimize any quantum simulation, not necessarily involving oracle calls, in which the linear combination rule is used to simulate a collection of constituent Hamiltonians. A straightforward extension of the analysis in Chapter 6 shows that the performance gain that would be achieved through the use of this technique is inversely proportional to the number of constituent Hamiltonians of the simulated system, and the quantum random walk of [3], where this number is only three, is a particularly suitable example for demonstrating a significant gain.

The sequence of ideas presented in this work is naturally tied to the question “What other problems can be solved exponentially faster on quantum computers than on classical ones?” More specifically, we can ask for what other problems quantum random walks can provide exponential algorithmic speedup. It is yet unknown whether this sort of questions can take us to a family of problems in speeding up the solution of which quantum random walks play a role similar to that of quantum Fourier transformation in case of solving hidden subgroup problems. However, it can be expected that there may be other families of graphs on which similar results can be shown. The variety of ideas that can be abstracted in graph formalism encourages us about the existence of useful computational problems which can efficiently be solved with use of the methods discussed here, or some modification of them.

REFERENCES

1. Grover, L., "A Fast Quantum Mechanical Algorithm for Database Search", *Proceedings of the 28th Annual ACM Symposium on Theory of Computing*, pp 212-219, 1996.

2. Shor, P. W., "Algorithms for Quantum Computing: Discrete Logarithm and Factoring", *Proceedings of 35th Annual Symposium on Foundations of Computer Science*, pp. 124-134, 1994.

3. Childs, A.M., R. Cleve, E. Deotto, E. Farhi, S. Gutmann and D.A. Spielman, "Exponential Algorithmic Speedup by Quantum Walk", *Procedings 35th ACM Symposium on Theory of Computing*, pp. 59-68, 2003.

4. Nielsen, M. A. and I. L. Chuang, *Quantum Computation and Quantum Information*, Cambridge University Press, 2000.

5. Hardy, Y. and W. H. Steeb, *Classical and Quantum Computing*, Birkhauser Verlag, Basel, Boston, Berlin, 2001.

6. Hirvensalo, M., *Quantum Computing*, Springer Verlag, Berlin, 2001.

7. Lomonaco, S. J., "A Rosetta Stone for Quantum Mechanics with an Introduction to Quantum Computation", *AMS Proceedings of Symposia in Applied Mathematics*, Vol. 58, pp 3 - 65, 2002.

8. Strang, G., *Linear Algebra and Its Applications,* Academic Press, New York, 1976.

9. Gruska, J., *Quantum Computing*, Mc Graw Hill, 1999.

10. Bennett, C. H., "Logical Reversibility of Computation" IBM Journal of Research and Development, Vol 17, pp. 525-532, 1973.

11. Barenco, A., D. Deutsch and A. Ekert, *Universality in Quantum Computation,* http://arxiv.org/pdf/quant-ph/9505018, 1995.

12. d'Espagnat, B., *Conceptual Foundations of Quantum Mechanics*, Perseus Books, Reading, Massachusetts, 1999.

13. Deutsch, D., "Quantum Theory, the Church-Turing Principle and the Universal Quantum Computer", *Proceedings of Royal Society of London*, Vol. A 400, pp. 97-117, 1985.

14. Bernstein, E. and U. Vazirani, "Quantum Complexity Theory", *Proceedings of the 25th Annual ACM Symposium on Theory of Computing*, pp 11 - 20, 1993.

15. Vazirani, U., "A Survey of Quantum Complexity Theory", *AMS Proceedings of Symposia in Applied Mathematics*, Vol. 58, pp 193-217, 2002.

16. Yao, A., "Quantum Circuit Complexity", *Proceedings of the 34th Annual IEEE Symposium on Foundations of Computer Science*, pp 352 - 360, 1993.

17. Aharonov, D., W. van Dam, J. Kempe, Z. Landau, S. Lloyd and O. Regev, *Adiabatic Quantum Computation is Equivalent to Standard Quantum Computation,* http://arxiv.org/pdf/quant-ph/0405098, 2005.

18. Bernstein, E. and U. Vazirani, "Quantum Complexity Theory", *SIAM Journal of Computing*, Vol. 26, No.5, pp. 1411-1473, 1997.

19. Kitaev, A., A. H. Shen and M. N. Vyalyi, *Classical and Quantum Computation*, volume 47 of Graduate Studies in Mathematics, American Mathematical Society (AMS), Providence, 2002.

20. Fortnow, L., "One Complexity Theorist's View of Quantum Computing", *Theoretical Computer Science*, Vol. 292, pp. 597-610, 2003.

21. Brassard, G., P. Høyer, M. Mosca and A. Tapp, *Quantum Amplitude Amplification and Estimation*, http://arxiv.org/pdf/quant-ph/0005055, 2000.

22. Deutsch, D. and R. Jozsa, "Rapid Solution of Problems by Quantum Computation", *Proceedings of Royal Society of London*. Vol. 439A, pp. 439-553, 1992.

23. Simon, D. R., "On the Power of Quantum Computation", *Proceedings of 35th Annual Symposium on Foundations of Computer Science*, pp. 116-123, 1994.

24. Shor, P. W., *Introduction to Quantum Algorithms*, http://arxiv.org/pdf/quant-ph/0005003, 2000.

25. Lomonaco, S. J. and L. H. Kauffman, *Quantum Hidden Subgroup Problems: A Mathematical Perspective*, http://arxiv.org/pdf/quant-ph/0201095, 2002.

26. Jozsa, R., *Quantum Factoring, Discrete Logarithms and the Hidden Subgroup Problem,* http://arxiv.org/pdf/quant-ph/0012084, 2000.

27. Pittenger, A. O., *An Introduction to Quantum Computing Algorithms*, Birkhauser Verlag, Boston, Basel, Berlin, 2001.

28. Brassard, G. and P. Høyer, *An Exact Quantum Polynomial-Time Algorithm for Simon's Problem,* http://arxiv.org/pdf/quant-ph/9704027, 1997.

29. Lomonaco, S. J., “Grover's Quantum Search Algorithm”, *AMS Proceedings of Symposia in Applied Mathematics*, Vol. 58, pp 181 - 192, 2002.

30. Bennett, H.C., E. Bernstein, G. Brassard and U. Vazirani, “Strengths and Weaknesses of Quantum Computing”, *SIAM Journal of Computing*, Vol. 26, No.5, pp. 1510-1523, 1997.

31. Apostol, T. M., *Introduction to Analytic Number Theory*, Springer-Verlag, New York, 1986.

32. Baker A., *A Concise Introduction to the Theory of Numbers*, Cambridge University Press, Cambridge, 1994.

33. Brandt, H. E., “Qubit Devices”, *AMS Proceedings of Symposia in Applied Mathematics*, Vol. 58, pp 67-139, 2002.

34. Lomonaco, S. J. (editor), *Quantum Computation: A Grand Mathematical Challenge for the Twenty-First Century and the Millennium*, volume 58 of Proceedings of Symposia in Applied Mathematics, American Mathematical Society (AMS), Providence, 2002.

35. Feynman, R.P., “Simulating physics with computers,” *International Journal of Theoretical Physics* Vol. 21, pp. 467–488, 1982.

36. Lloyd, S., "Universal quantum simulators", *Science,* Vol. 273, pp. 1073–1078, 1996.

37. Wiesner, S., *Simulations of Many Body Quantum Systems by a Quantum Computer*, http://arxiv.org/pdf/quant-ph/9603028, 1996.

38. Zalka, C., *Efficient Simulations of Quantum Systems by Quantum Computer*, http://arxiv.org/pdf/quant-ph/9603026, 1996.

39. Childs, A, M., *Quantum Information Processing in Continuous Time*, (Doctoral Dissertation, Massachusetts Institute of Technology), http://www.cs.caltech.edu/~amchilds/thesis.pdf, 2004.

40. Farhi, E., J. Goldstone, S. Gutmann and M. Sipser, *Quantum Computation by Adiabatic Evolution*, http://arxiv.org/pdf/quant-ph/0001106, 2000.

41. Aharonov, D., A. Ambainis, J. Kempe and U. Vazirani, *Quantum Walks on Graphs,* (version 2), http://arxiv.org/pdf/quant-ph/0012090, 2002.

42. Kempe, J., *Quantum Random Walks – an Introductory Overview*, http://arxiv.org/pdf/quant-ph/0303081, 2003.

43. Kempe, J., *Quantum Walks Hit Exponentially Faster*, http://arxiv.org/pdf/quant-ph/0205083, 2002.

44. Farhi, E., S. Gutmann, *Quantum Computation and Decision Trees*, (version 2), http://arxiv.org/pdf/quant-ph/9706062, 1998.

45. Ambainis, A., *Quantum Walks and Their Algorithnic Applications*, (version 3), http://arxiv.org/pdf/quant-ph/0403120, 2005.

46. Ambainis, A., J. Kempe, A. Rivosh, *Coins Make Quantum Walks Faster*, http://arxiv.org/pdf/quant-ph/0402107, 2004.

47. Childs, A. M., E. Farhi, S.Gutmann, *An Example of the Difference Between Quantum and Classical Random Walks*, http://arxiv.org/pdf/quant-ph/0103020, 2001.

48. Shenvi, N., J.Kempe, K.Whaley, *Quanrum Random Walk Search Algorithm*, http://arxiv.org/pdf/quant-ph/0210064, 2005.

49. Childs, A. M., J. Goldstone, *Spatial Search by Quantum Walk*, (version 2), http://arxiv.org/pdf/quant-ph/0306054, 2004.

50. Tregenna, B., W. Flanagan, R. Maile and V. Kendon, *Controlling Discrete Time Quantum Walks: Coins and Initial States*, (version 2), http://arxiv.org/pdf/quant-ph/0304204, 2003.