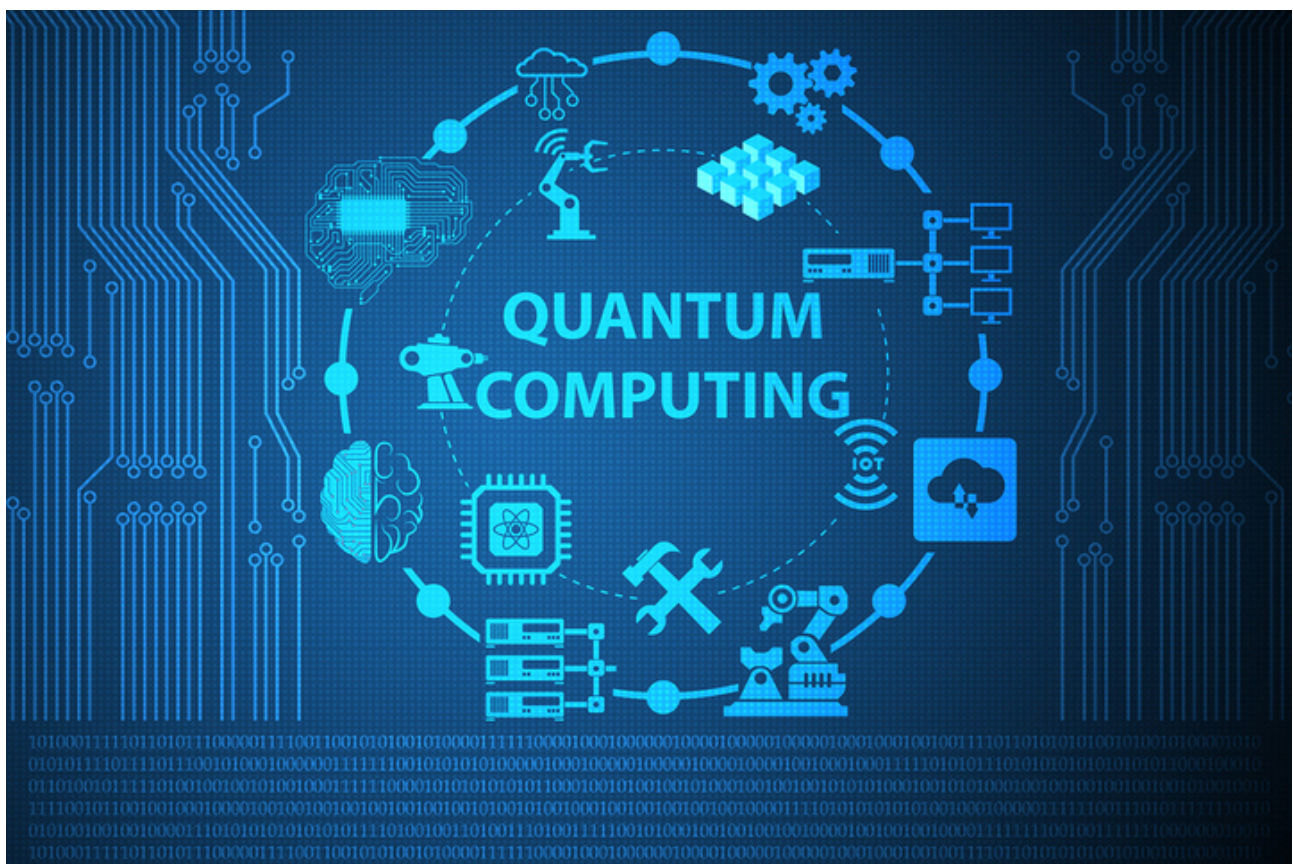


SUMMER OF SCIENCE

FULL-TERM REPORT

Introduction To Quantum Computing & Algorithms



HARSHIT GUPTA

Mentor: Neeraj Sohani

Roll No.: 190050048

Department of Computer Science and Engineering

Contents

I	Quantum Circuits	3
1	Classical Ideas	4
1.1	Introduction	4
1.2	Turing Machines	5
1.2.1	Universal Turing Machine	6
1.2.2	Church Turing Thesis	6
1.2.3	Halting Problem	6
1.3	Circuits	8
1.3.1	Circuit Families	9
1.3.2	Computational Complexity	10
1.3.3	Complexity Classes	10
2	Basics of Quantum Mechanics	13
2.1	Introduction	13
2.2	Postulates of Quantum Mechanics	14
2.2.1	State Space	14
2.2.2	Evolution	14
2.2.3	Quantum Measurement	15
2.2.4	Composite Systems	16
2.3	Another Way to Define Quantum Systems - Density Operator	17
2.3.1	Postulates of Quantum Mechanics in terms of Density Operator	18
2.3.2	Reduced Density Operator	18
2.3.3	Schmidt Decomposition and Purification	19
3	Quantum Circuits	20
3.1	Introduction	20
3.2	Qubits	20
3.2.1	Bloch Sphere	20
3.3	Single Qubit Operations	21
3.4	Controlled Operation	22
3.5	Measurement	24
3.6	Universal Quantum gates	25
3.6.1	Two-level gates are Universal	25
3.6.2	Single Qubit gates and CNOT are universal	26
3.6.3	Approximating Single-Qubit Gates	26
3.7	Solovay-Kitaev Theorem	28
3.7.1	Proof of Solovay Kitaev theorem	28

II	Algorithms	30
4	Deutsch-Josza Algorithm	31
4.1	Problem	31
4.2	Classical Solution	31
4.3	Quantum Solution	31
5	Simon's Algorithm	33
5.1	Problem	33
5.2	Classical Soution	33
5.3	Quantum Solution	33
6	Quantum Fourier Transform	35
6.1	Fourier Transform	35
6.2	Classical Solution	35
6.3	Quantum Solution	36
7	Applications of Quantum Fourier Transform	39
7.1	Quantum Phase Estimation	39
7.1.1	Problem	39
7.1.2	Solution	39
7.2	Order-finding	42
7.2.1	Problem	42
7.2.2	Classical Solution	42
7.2.3	Quantum Solution	42
7.3	Period Finding(Shor's Algorithm)	44
7.3.1	Problem	44
7.3.2	Classical Solution	44
7.3.3	Quantum Solution	44
7.4	Factorization	46
7.4.1	Problem	46
7.4.2	Classical Solution	46
7.4.3	Quantum Solution	46
8	Quantum Search	48
8.0.1	Oracle	48
8.0.2	Solution(Grover's Algorithm)	48

Part I

Quantum Circuits

Chapter 1

Classical Ideas

1.1 Introduction

To learn quantum phenomena, it is quite helpful to first learn the classical constructs and techniques already available as various things in quantum are built as an analogy to the classical case. This chapter introduces the basic concepts in classical computing. I will start with the Turing machine model and circuit model for computation and establish the relationship between these two models. Then I will move on to the computational complexity analysis, a field which examines the time and space requirements necessary to solve particular computational problems, and provides a broad classification of problems based upon their difficulty of solution.

1.2 Turing Machines

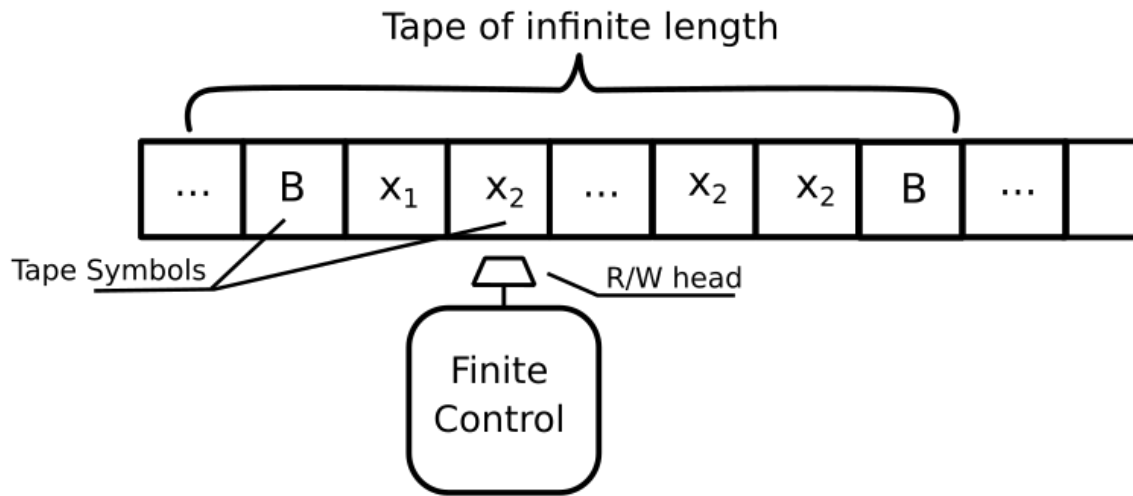


Figure 1.1: A Turing Machine

A Turing machine contains four main elements:

- (a) a program, rather like an ordinary computer which tells what to do for a given instruction;
- (b) a finite state control, which acts like a stripped-down microprocessor, co-ordinating the other operations of the machine;
- (c) a tape, which acts like a computer memory; and
- (d) a read- write tape-head, which points to the position on the tape which is currently readable or writable.

The finite state control for a Turing machine consists of a finite set of internal states, q_1, \dots, q_m . The number m is allowed to be varied; it turns out that for m sufficiently large this does not affect the power of the machine in any essential way, so without loss of generality we may suppose that m is some fixed constant. Along with these m states, there are two more states, the starting state q_s and the halting state q_h which denotes the starting and the end of the process respectively. This means that when the processing starts machine will be in the starting state, and when the process is completed, the state control changes to the halting state. These states can be thought of as the internal storage of the machine which helps in the processing.

The Turing machine tape is a one-dimensional object, which stretches off to infinity in one direction. The tape consists of an infinite sequence of tape squares numbered $0, 1, 2, \dots$. The tape squares each contain one symbol drawn from some alphabet, Γ , which contains a finite number of distinct symbols. The first symbol is always fixed (say \triangleright) denoting the left edge(starting) of the tape.

The tape-head points at a single square in the tape and is used to read the symbol from that location and write new information.

A program is a finite set of orders that are to be executed given an internal state and symbol on the tape. So basically, it's a mapping from $\langle q, x \rangle$ to $\langle q', x', s \rangle$ where q and x are the initial internal state and tape square symbol and q' and x' are the final states and symbol. s defines the movement of the tape-head i.e. after updating the state and symbol where the head should go. It can be 0 (stay on the same location), 1 (move right) or -1 (move left except if on the leftmost square). The commands are generally written as $\langle q, x, q', x', s \rangle$ were symbols means the same as before. So, a command of $\langle q_1, 1, q_2, 2, 1 \rangle$ will mean that if we encounter

the symbol 1 when the internal state is q_1 , change the state to q_2 and overwrite the symbol to 2. Then move the cursor one square towards the right.

A lot of modifications are possible in the turing machines. For e.g.: There may be two tapes instead of one in the machine. These modifications may simplify the algorithm used but they don't offer much computational advantage and can actually be shown to be equivalent to the normal turing machines. Hence, we consider only the simple turing machine as described above.

1.2.1 Universal Turing Machine

In a given turing machine, we can vary the program, the internal states and the contents of the tape. But it turns out that we can use a single machine called the *Universal Turing Machine* in place of the all the turing machines.

A universal turing machine has a fixed internal state and program. So, the only thing that can vary is the content of the tape. In universal turing machine, we first "tell" the machine which machine it must act as and then the input. These all things, the turing machine details and input for that machine, is given through the tape. The turing machine is specified using the Turing number. A Turing number is a unique number associated with each turing machine.

This is very much similar to a programmable computer. All the commands are already stored in the memory and the user has to just tell which program to execute and the inputs for that program. The idea of the universal turing machine allows us to think of a real machine which can compute any given function (computable by algorithm)

1.2.2 Church Turing Thesis

It turns out that this simple model can be used to compute any function which has a definite algorithm. It is stated officially as the *Church-Turing Thesis* :

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

This theorem is very important as it defines what all functions can be computed by a classical computer and by quantum computers too!! Though it is not yet clear that what it means to say "computable by algorithm" and hence there may exist another computational method which may compute a function not computable by turing machines.

If we allow randomness too in the model, we get a probabilistic Turing Machine. And hence we get the *Strong Church-Turing Thesis* :

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

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

1.2.3 Halting Problem

This problem is a modification of the Hilbert *entscheidungsproblem* which asks whether it is possible to compute all the problems in mathematics. The halting problem is

HALTING PROBLEM: Does the machine with Turing number x halt upon input of the number y

This can easily be seen that the halting problem is equivalent to the Hilbert *entscheidungsproblem*.

Turing proved that it is not possible on a sub-case where the input is same as turing number and hence answered the Hilbert *entscheidungsproblem* with a negation. To show this, he defined a halting function:

$$h(x) = \begin{cases} 0 & \text{if machine number } x \text{ does not halt upon input of } x \\ 1 & \text{if machine number } x \text{ halts upon input of } x \end{cases} \quad (1.1)$$

If there is an algorithm to solve the halting problem, then there is one to compute $h(x)$. We will show that an algorithm to compute $h(x)$ can't exist. Consider the turing machine running the following pseudo-code:

```
y = h(x)
if y=0
    then halt
else
    loop forever
```

Clearly the algorithm is completely defined and hence it is possible to make the turing machine. But, if we give this turing machine its own number, we can easily see that we arrive at a contradiction. Hence, $h(x)$ can't be calculated and so, the answer to *entscheidungsproblem* is a clear **NO**.

1.3 Circuits

Turing machines are rather idealized models of computing devices. Real computers are finite in size, whereas for Turing machines we assumed a computer of unbounded size. The Circuits model provides a more realistic way of making a real computer.

A circuit is made up of wires and gates, which carry information around, and perform simple computational tasks, respectively. A circuit may involve multiple input and output bits, many wires, and many logical gates. A logic gate is a function $f : \{0, 1\}^k \mapsto \{0, 1\}^l$ from some fixed number k of input bits to some fixed number l of output bits.

The following image shows the standard symbols for each of the elementary gates along with there truth tables.

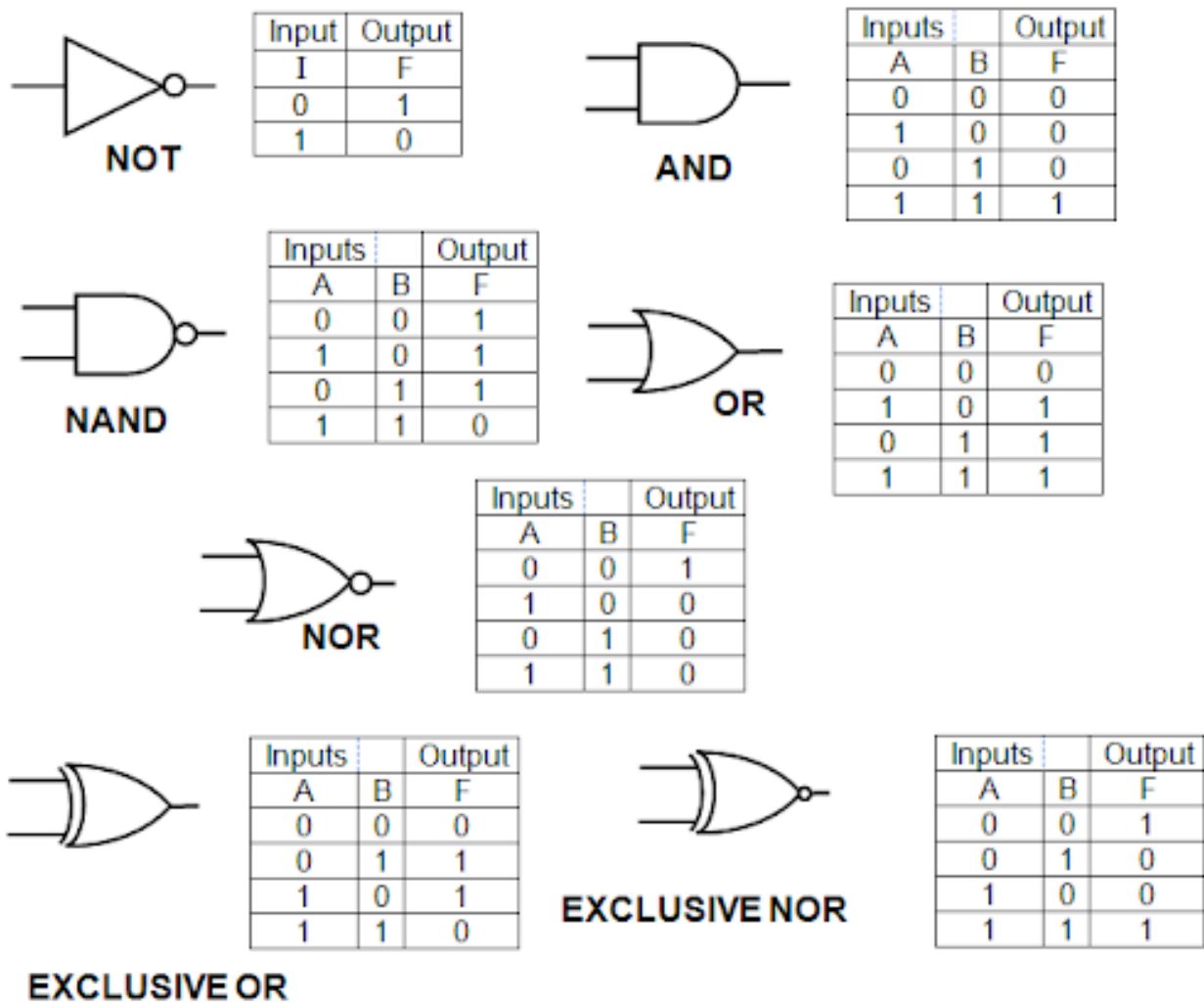


Figure 1.2: Elementary gates

Along with these gates there are two more gates, FANOUT and CROSSOVER. FANOUT is gate used to duplicate a bit or rather 'divide' a bit into two with same value. CROSSOVER is used to interchange the values of two bits.

These basic gates can be combined to do a large number of operations. Actually, these gates can be used to compute any function $f: \{0, 1\}^n \mapsto \{0, 1\}^n$. Moreover, among these elementary gates, NAND and NOR are called **Universal Gates** as they can be used to simulate other elementary gates (given wires, ancilla and FANOUT is available.) As an example the image

below shows the circuit used to add two binary numbers:

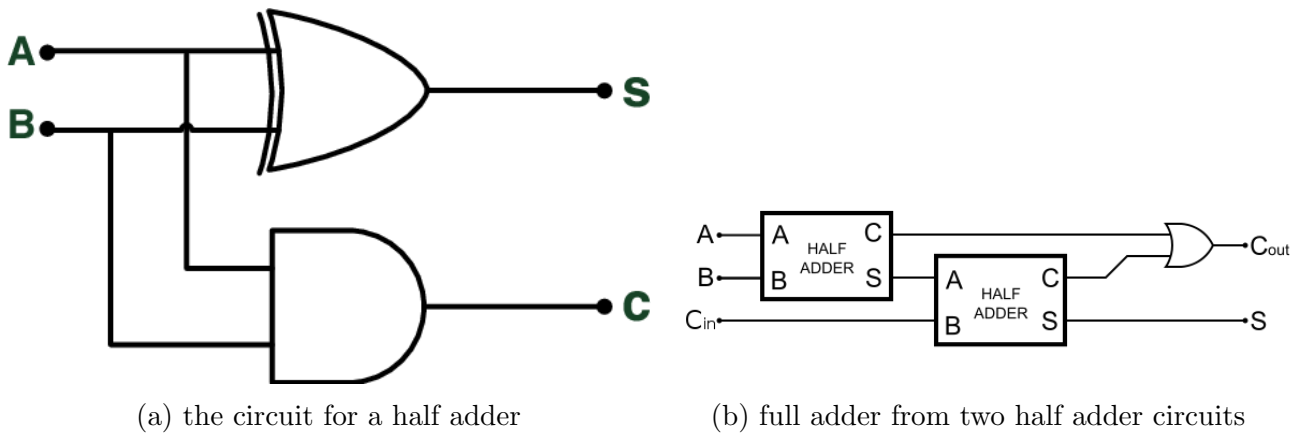


Figure 1.3: Circuit to add two binary numbers

1.3.1 Circuit Families

A **Circuit Family** consists of a collection of circuits, $\{C_n\}$, indexed by a positive integer n . The circuit C_n has n input bits, and may have any finite number of extra work bits, and output bits. The output of the circuit C_n , upon input of a number x of at most n bits in length, is denoted by $C_n(x)$. We require that the circuits be consistent, that is, if $m < n$ and x is at most m bits in length, then $C_m(x) = C_n(x)$. The function computed by the circuit family $\{C_n\}$ is the function $C(\cdot)$ such that if x is n bits in length then $C(x) = C_n(x)$.

A family of circuits $\{C_n\}$ is said to be a **uniform circuit family** if there is some algorithm running on a Turing machine which, upon input of n , generates a description of C_n . That is, the algorithm outputs a description of what gates are in the circuit C_n , how those gates are connected together to form a circuit, any ancilla bits needed by the circuit, FANOUT and CROSSOVER operations, and where the output from the circuit should be read out. In simple terms, there must exist a definite algorithmic way in which an engineer can construct a circuit for $\{C_n\}$ for any n .

With this uniformity restriction, results in the Turing machine model of computation can usually be given a straightforward translation into the circuit model of computation, and vice versa.

1.3.2 Computational Complexity

Computational complexity is the study of the time and space resources required to solve computational problems. The task of computational complexity is to prove lower bounds on the resources required by the best possible algorithm for solving a problem, even if that algorithm is not explicitly known.

The chief distinction made in computational complexity is between problems which can be solved using resources which are bounded by a polynomial in n , or which require resources which grow faster than any polynomial in n . In the latter case we usually say that the resources required are *exponential* in the problem size.

The 'polynomial increase' mentioned in *Strong Church-Turing Thesis* is this polynomial class only. So, the strong Church Turing Thesis says that any algorithm which can be run on some mode of computation can be run on the probabilistic Turing machine in $p(k)$ where k is number of elementary operations required in that mode of computation and $p(\cdot)$ is a polynomial function.

By *Strong Church-Turing Thesis*, we can say that if an algorithm can not be run(in polynomial time) on the probabilistic Turing machine, then it can't be run on any machine. Thus, the strong Church-Turing thesis implies that the entire theory of computational complexity will take on an elegant, model-independent form if the notion of efficiency is identified with polynomial resource algorithms, and this elegance has provided a strong impetus towards acceptance of the identification of 'solvable with polynomial resources' and 'efficiently solvable'.

1.3.3 Complexity Classes

A **Complexity Class** contains a set of problems that take a similar range of space and time to solve, for example "all problems solvable in polynomial time with respect to input size," "all problems solvable with exponential space with respect to input size," and so on. Some complexity classes may be a subset of others. The following figure shows various complexity classes along with their relation (Some relations are not yet proved. E.g.: it is still unknown whether $NP = P$ or not).

P Class

The class P contains decision problems (problems with a "yes" or "no" answer) that are solvable in polynomial time by a deterministic Turing machine. A problem in P can be solved in n^k time for some constant k and where n is the size of the input.

Some famous problems in P include:

- Finding if the given numbers are relatively prime (and in general prime too)
- Linear Programming
- Greatest Common Divisor

NP and co-NP Class

The class NP contains decision problems (problems with a "yes" or "no" answer) that are solvable by a Turing machine in non-deterministic polynomial time — this includes problems that are solvable in polynomial time up to problems that are solvable in exponential time. While they can take a long time to solve, problems in NP can be verified by a Turing machine in polynomial time. This means that if given a yes answer to an NP problem, you can check that it is right in polynomial time. This yes answer is often called a witness or a certificate.

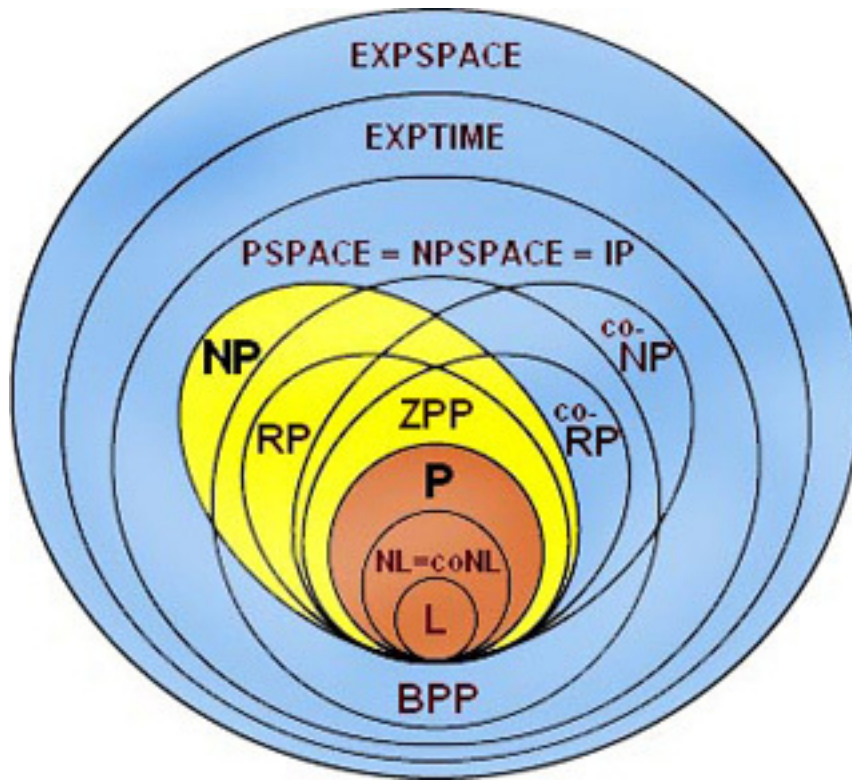


Figure 1.4: Various Complexity Classes

More rigorously, a language L is in NP if there is a Turing machine M with the following properties:

1. If $x \in L$ then there exists a witness string w such that M halts in the state q_Y (yes state) after a time polynomial in size of x when the machine is started in the state x -blank- w .
2. If $x \in L$ then for all strings w which attempt to play the role of a witness, the machine halts in state q_N (no state) after a time polynomial in size of x when M is started in the state x -blank- w .

Language of an algorithm/problem can be understood as the set of all the inputs which results in the “yes” state.

If a problem X is in NP , then its complement, \bar{X} is in $coNP$. $coNP$ contains problems that have a polynomial time verification for “no” answers — if given a solution that does not solve the problem, it is easy to verify if that solution does not work.

Relations between P , NP and $coNP$

$$P \subseteq NP$$

$$P \subseteq coNP$$

hence,

$$P = NP \cap coNP$$

It is still unknown whether $P = NP$. In such a case $P = NP = coNP$. If $P \neq NP$, it will lead to a new class, **NPI** (NP intermediate) which are not solvable in polynomial time but can be verified in polynomial time.

An important class of problems in NP class is the *NP-complete*.

NP-Complete Problems

NP-complete problems are very special because any problem in the *NP* class can be transformed or reduced into *NP-complete* problems in polynomial time. This means that if you can solve an *NP-complete* problem, you can solve any other problem in *NP*. An important consequence of this is that if you could solve an *NP-complete* problem in polynomial time, then you could solve any *NP* problem in polynomial time.

The famous **Cook-Levin Theorem** establishes the *NP-Completeness* of CSAT.

CSAT: Given a Boolean circuit composed of AND, OR and NOT gates, is there an assignment of values to the inputs to the circuit that results in the circuit outputting 1, that is, is the circuit satisfiable for some input?

All the other problems are shown to be *NP-Complete* by reducing CSAT or previously determined *NP-Complete* problem to that problem and using the *transitivity* of *reduction*.

Some famous *NP-Complete* problems are:

- Knapsack problem
- Graph isomorphism
- Hamiltonian Path Problem

PSPACE Class

The class PSPACE is the set of decision problems that can be solved by a deterministic Turing machine in a polynomial amount of space with respect to the input size.

Relations between P, NP and PSPACE:

$$P \subseteq PSPACE$$

$$NP \subseteq PSPACE$$

$$PSPACE \subseteq EXP$$

The final relation including the P, NP, PSPACE, L(*Logarithmic Space*) and EXP(*Exponential Time*) is:

$$L \subseteq P \subseteq NP \subseteq PSPACE \subseteq EXP$$

It is known that $P \neq EXP$ and $L \neq PSPACE$. So, at least one of the inclusions is strict.

Chapter 2

Basics of Quantum Mechanics

2.1 Introduction

This chapter explains the various concepts and notations needed to study quantum computing. The quantum computing requires a bit of linear algebra, so it is better to revise it up before starting with quantum mechanics. Then, I will move on to two different ways of dealing with quantum mechanics, by defining *the quantum state* or by defining *density operator*. The following table provides basic notations used.

Notations in linear algebra	
Notation	Description
z^*	Complex conjugate of the complex number z .
$ \psi\rangle$	Vector. Also known as a ket.
$\langle\psi $	Vector dual to $ \psi\rangle$. Also known as a bra.
$\langle\phi \psi\rangle$	Inner product between the vectors $ \phi\rangle$ and $ \psi\rangle$.
$ \phi\rangle \otimes \psi\rangle$	Tensor product of $ \phi\rangle$ and $ \psi\rangle$.
$ \phi\rangle \psi\rangle$	Abbreviated notation for tensor product of $ \phi\rangle$ and $ \psi\rangle$.
A^*	Complex conjugate of the matrix A .
A^T	Tranpose of the matrix A .
A^\dagger	Hermitian conjugate of the matrix A .
$\langle\phi A \psi\rangle$	Inner product between the vectors $ \phi\rangle$ and $A \psi\rangle$.
$ \psi\rangle $	Norm of the vector $ \psi\rangle = \sqrt{\langle\psi \psi\rangle}$

2.2 Postulates of Quantum Mechanics

Quantum mechanics is a mathematical framework for the development of physical theories. On its own quantum mechanics doesn't tell you what laws a physical system must obey, but it does provide a mathematical and conceptual framework for the development of such laws.

2.2.1 State Space

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

This postulate merely tells us that there will be one which will describe the system, but in general, it is very hard to find the state the system is actually in.

Since the state space is a Hilbert space, it will have some basis vectors. So, we can express the state of the system in terms of basis vectors. The most simplest quantum mechanical system is a qubit which has 2 orthonormal basis states generally denoted by $|0\rangle$ and $|1\rangle$ (in analogy to the bits in the classical computer). So, any state can be written as

$$|\psi\rangle = a|0\rangle + b|1\rangle \quad (2.1)$$

where a and b are complex numbers such that $|\psi\rangle$ is a unit vector. The qubits play the role of bits in a quantum computer.

2.2.2 Evolution

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

$$|\psi'\rangle = U|\psi\rangle \quad (2.2)$$

Similar to the states, quantum mechanics does not tell the U matrix, it just says that it will be a unitary matrix.

In quantum computing, we use qubits and hence any computation must be an evolution of qubits only. This means that operations allowed on a qubit (so that they may be realized in a real system) are Unitary operators. Moreover, in case of single qubits, it turns out that any unitary operator at all can be realized in realistic systems.

The postulate 2 can be re framed in terms of differential equations as the famous **Schrödinger Equation**:

Postulate 2': The time evolution of the state of a closed quantum system is described by the Schrödinger equation,

$$i\hbar \frac{d|\psi\rangle}{dt} = H|\psi\rangle. \quad (2.3)$$

In this equation, \hbar is a physical constant known as reduced Planck's constant whose value is determined as $1.054571817 \times 10^{-34}$. H is a fixed Hermitian operator known as the Hamiltonian of the closed system. // If we know the Hamiltonian, we, at least in principle, know the dynamics of the system completely. Also, since Hamiltonian is hermitian, it has a spectral decomposition:

$$H = \sum_E E|E\rangle\langle E| \quad (2.4)$$

Here, E are the eigenvalues and $|E\rangle$ are the eigenvectors also known as stationary states. The value E denotes the Energy of energy state $|E\rangle$. They are called as stationary states because if the system is present in one of these states, then the system does not change its state (upto a non-significant numerical factor).

Postulate 2' can be reduced to Postulate 2 by solving the differential equation:

$$|\psi'\rangle = \exp\left[\frac{-iH(t_2 - t_1)}{\hbar}\right]|\psi\rangle = U(t_2, t_1)|\psi\rangle \quad (2.5)$$

(In this equation, we have taken exponentiation of a matrix. The function $f(A)$ of a normal matrix A is defined as: $f(A) = \sum_i f(\lambda_i)|e_i\rangle\langle e_i|$ where λ_i are eigenvalues of A and $|e_i\rangle$ are eigenvectors of A).

2.2.3 Quantum Measurement

Postulate 3: Quantum measurements are described by a collection $\{M_m\}$ of measurement operators. These are operators acting on the state space of the system being measured. The index m refers to the measurement outcomes that may occur in the experiment. If the state of the quantum system is $|\psi\rangle$ immediately before the measurement then the probability that result m occurs is given by

$$p(m) = \langle\psi|M_m^\dagger M_m|\psi\rangle \quad (2.6)$$

and the state of the system after measurement is

$$\frac{M_m|\psi\rangle}{\sqrt{\langle\psi|M_m^\dagger M_m|\psi\rangle}} \quad (2.7)$$

Since the probabilities must sum to 1, we get the *completeness equation*:

$$\sum_m M_m^\dagger M_m = I \quad (2.8)$$

An important class of measurements are the *projective measurements*:

Projective measurements: A projective measurement is described by an observable, M , a Hermitian operator on the state space of the system being observed. The observable has a spectral decomposition,

$$M = \sum_m m P_m \quad (2.9)$$

where P_m is the projector on the eigenspace of M with eigenvalue m . By postulate we can easily see that

$$p(m) = \langle\psi|P_m|\psi\rangle \text{ and } |\psi'\rangle = \frac{P_m|\psi\rangle}{\sqrt{p(m)}} \quad (2.10)$$

In addition to this, we can calculate the average value and standard deviation of the measurement as

$$\begin{aligned} E(M^k) &= \sum_m m^k p(m) \\ &= \sum_m m^k \langle\psi|P_m|\psi\rangle \\ &= \langle\psi| \left(\sum_m m^k P_m \right) |\psi\rangle \\ &= \langle\psi|M^k|\psi\rangle \end{aligned} \quad (2.11)$$

noting that standard deviation can be calculated as :

$$|\Delta(M)|^2 = E(M)^2 - E(M^2)$$

This formulation of measurement and standard deviations in terms of observables gives rise in an elegant way to results such as the [Heisenberg uncertainty principle](#).

An interesting result is that any measurement can be written in the form of product of a Unitary operator and projective measurement. It is easier to show this result using Composite system and hence we discuss it now.

2.2.4 Composite Systems

Postulate 4: The state space of a composite physical system is the tensor product of the state spaces of the component physical systems. Moreover, if we have systems numbered 1 through n , and system number i is prepared in the state $|\psi_i\rangle$, then the joint state of the total system is $|\psi_1\rangle \otimes |\psi_2\rangle \otimes \dots \otimes |\psi_n\rangle$.

Let us now prove the result that any measurement can be written as the product of Unitary matrix and projective measurement. Let us suppose that the measurement is M_m upon the state space \mathcal{Q} . Let there be another state space \mathcal{M} having orthonormal basis $|m\rangle$ in correspondence with the outcomes of M_m .

Let us define a unitary operator U such that

$$U|\psi\rangle|0\rangle = \sum_m M_m|\psi\rangle|m\rangle \quad (2.12)$$

Clearly there will be a unitary operator satisfying this property. Now consider the projective measurement $P_m \equiv I_{\mathcal{Q}} \otimes |m\rangle\langle m|$. Then,

$$\begin{aligned} P_m U|\psi\rangle|0\rangle &= \sum_{m'} (I_{\mathcal{Q}} \otimes |m\rangle\langle m|) M_{m'} |\psi\rangle|m'\rangle \\ &= M_m |\psi\rangle \end{aligned} \quad (2.13)$$

Using $M_m|\psi\rangle$ as given by the previous equation, we can easily express $p(m)$ and state after measurement in terms of U and P_m . Hence, we can reduce any measurement into the product of a Unitary operator and projective measurement.

Another important consequence of Postulate 4 is *entanglement*. This means that the state of a composite system can't be reduced into a product of states of individual systems and hence the state of one system is "dependent" on the state of the other system. For e.g.

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

In this, the state of both the individual system will be same i.e. if we measure both the systems, then the outcome will be same. There are a lot of useful consequences and applications of entanglement including [Quantum Teleportation](#) and [Superdense Coding](#).

2.3 Another Way to Define Quantum Systems - Density Operator

The density operator is used to define a quantum system whose states are not exactly known. If we have the quantum system in one of the states $|\psi_i\rangle$ with probability p_i , then *density operator* is defined as :

$$\rho = \sum_i p_i |\psi_i\rangle \langle \psi_i| \quad (2.14)$$

The pair $\{p_i, |\psi_i\rangle\}$ is known as an ensemble of pure states.

It turn out that the postulates of quantum mechanics can be re-written in the form of density operator. Hence, the density operator representation of a quantum system is complete.

Since, the density operator representation of a system is complete, we must be able to define density operator without using the state vectors. We can define the density operator as follows :

An operator ρ is the density operator associated to some ensemble $\{p_i, |\psi_i\rangle\}$ if and only if it satisfies the following condition :

1. ρ has a unit trace
2. ρ is a positive operator (a hermitian operator with positive eigenvalues.)

The proof of the above statement is easy to see. Since ρ is a positive operator, it has a spectral decomposition as

$$\rho = \sum_i \lambda_i |i\rangle \langle i| \quad (2.15)$$

comparing it with the ensemble definition it turns out that $p_i = \lambda_i$ and $|\psi_i\rangle = |i\rangle$ is a valid ensemble definition corresponding to the density operator ρ . The first condition is required so as to meet the total probability criteria i.e.

$$\begin{aligned} \sum_i p_i &= 1 \\ \sum_i \lambda_i &= 1 \\ \text{Trace}(\rho) &= 1 \end{aligned} \quad (2.16)$$

Along with this, if the system is in pure state (only one of the states is possible), then

$$\text{Trace}(\rho_{\text{pure}}^2) = 1 \quad (2.17)$$

If the system is not in *pure state* then it is said to be in *mixed state*.

An important point to remember is that the ensemble corresponding to a density operator is not unique. This means that two different ensembles may have the same density operator. If two ensemble say $\{p_i, |\psi_i\rangle\}$ and $\{q_j, |\phi_j\rangle\}$ has the same density operator, then

$$\sqrt{p_i} |\psi_i\rangle = \sum_j u_{i,j} \sqrt{q_j} |\phi_j\rangle \quad (2.18)$$

where $u_{i,j}$ is a unitary matrix of complex numbers, with indices i and j . The converse of the above statement is also true.

2.3.1 Postulates of Quantum Mechanics in terms of Density Operator

Postulate 1: Associated to any isolated physical system is a complex vector space with inner product (that is, a Hilbert space) known as the state space of the system. The system is completely described by its density operator, which is a positive operator ρ with trace one, acting on the state space of the system. If a quantum system is in the state ρ_i with probability p_i , then the density operator for the system is $\sum_i p_i \rho_i$.

Postulate 2: The evolution of a closed quantum system is described by a unitary transformation. That is, the state ρ of the system at time t_1 is related to the state ρ' of the system at time t_2 by a unitary operator U which depends only on the times t_1 and t_2 ,

$$\rho' = U \rho U^\dagger \quad (2.19)$$

Postulate 3: Quantum measurements are described by a collection $\{M_m\}$ of *measurement operators*. These are operators acting on the state space of the system being measured. The index m refers to the measurement outcomes that may occur in the experiment. If the state of the quantum system is ρ immediately before the measurement then the probability that result m occurs is given by

$$p(m) = \text{tr}(M_m^\dagger M_m \rho) \quad (2.20)$$

and the state of system after measurement is given by

$$\rho' = \frac{M_m \rho M_m^\dagger}{\text{tr}(M_m^\dagger M_m \rho)} \quad (2.21)$$

The measurement operator satisfies the *completeness equation*:

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

Postulate 4: The state space of a composite physical system is the tensor product of the state spaces of the component physical systems. Moreover, if we have systems numbered 1 through n , and system number i is prepared in the state ρ_i , then the joint state of the total system is $\rho_1 \otimes \rho_2 \otimes \dots \otimes \rho_n$.

2.3.2 Reduced Density Operator

Suppose we have physical systems A and B, whose state is described by a density operator ρ^{AB} . The reduced density operator for system A is defined by

$$\rho^A \equiv \text{tr}_B(\rho^{AB}) \quad (2.22)$$

where tr_B is the partial trace defined by

$$\text{tr}_B(|a_1\rangle\langle a_2| \otimes |b_1\rangle\langle b_2|) = |a_1\rangle\langle a_2| \cdot \text{tr}(|b_1\rangle\langle b_2|) = \langle b_1|b_2\rangle |a_1\rangle\langle a_2| \quad (2.23)$$

The reduced density operator is so useful as to be virtually indispensable in the analysis of composite quantum systems.

2.3.3 Schmidt Decomposition and Purification

Schmidt Decomposition : Suppose $|\psi\rangle$ is a pure state of a composite system, AB . Then, there exist orthonormal basis $|i_A\rangle$ for system A and orthonormal basis $|i_B\rangle$ of system B such that

$$|\psi\rangle = \sum_i \lambda_i |i_A\rangle |i_B\rangle \quad (2.24)$$

where λ_i are non-negative real numbers satisfying $\sum_i \lambda_i = 1$ known as *Schmidt Coefficients*. The basis $|i_A\rangle$ and $|i_B\rangle$ are called the *Schmidt bases* for A and B and the number of λ_i is called the *Schmidt Number* for the state $|\psi\rangle$.

The Schmidt Decomposition is essentially a restatement of Singular Value Decomposition applied on the coordinate matrix in the standard basis of A and B . (Co-ordinate matrix = $\{c_{i,j}\}$ where $c_{i,j}$ is the coefficient of $|i\rangle \otimes |j\rangle$ in the expansion of $|\psi\rangle$. Here, $|i\rangle$ and $|j\rangle$ are the standard basis of state space of A and B .)

Combining it with the reduced density operator, we get

$$\rho^A = \sum_i \lambda_i^2 |i_A\rangle \langle i_A| \text{ and } \rho^B = \sum_i \lambda_i^2 |i_B\rangle \langle i_B| \quad (2.25)$$

showing that the eigenvalues of ρ^A and ρ^B are same and equal to λ_i^2 . This leads to one another way to define a pure state apart from the trace definition.

Another related technique is *purification*. Suppose we are given a state ρ^A of a quantum system A . It is possible to introduce another system, which we denote R , and define a pure state $|AR\rangle$ for the joint system AR such that $\rho^A = \text{tr}_R(|AR\rangle \langle AR|)$. That is, the pure state $|AR\rangle$ reduces to ρ^A when we look at system A alone. This is a purely mathematical procedure, known as purification, which allows us to associate pure states with mixed states. For this reason we call system R a reference system: it is a fictitious system, without a direct physical significance.

The purification of ρ^A is done by finding the spectral decomposition of ρ^A and then taking tensor product each eigenvector with itself. So,

$$\text{if } \rho^A = \sum_i \lambda_i |i\rangle \langle i| \text{ then } |AR\rangle = \sum_i \sqrt{\lambda_i} |i^A\rangle |i^R\rangle \quad (2.26)$$

This $|AR\rangle$ can easily be seen to be a pure state by Schmidt Decomposition. In such a case, the state space of R will be same as that of A . This R is not unique and there will exist infinite number of R . An easy example would be by replacing $|i^R\rangle$ by $U|i^R\rangle$ where U is a unitary matrix. (This can be thought of as the transformation of the state space of R from $|i^R\rangle$ to another basis defined by the unitary matrix U .)

Chapter 3

Quantum Circuits

3.1 Introduction

This chapter onwards, we start with quantum computing. The basic entity used to store information in a quantum computer is a qubit and all the operations in a quantum computer can be decomposed into operations on these qubits. Then we move on to describe what all operations can be done on single as well as multiple qubits ending with a few algorithms to perform simple operations on a quantum computer.

3.2 Qubits

As defined earlier, qubits are quantum systems whose state space has cardinality of two. The standard orthonormal basis of qubits are denoted by $|0\rangle$ and $|1\rangle$ (also known as *standard computational basis*). So, any value of a qubit can be expressed as

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \quad (3.1)$$

where α and β are complex numbers such that $\| |\psi\rangle \| = 1$. As mentioned in the postulates of quantum mechanics, we can't directly measure α and β . So, the algorithms must be designed in such a way that it does not require explicit knowledge of α and β . So, although we may do various things to manipulate α and β but it is very difficult to retrieve that result from the result. This inability restricts us from using the immense power we get from having infinite states (as compared to the two states in the classical bits).

3.2.1 Bloch Sphere

From the normal condition, we get $\alpha^2 + \beta^2 = 1$. So, we may re-write $\alpha = e^{i\gamma} \cos \frac{\theta}{2}$ and $\beta = e^{i(\gamma+\phi)} \sin \frac{\theta}{2}$. So, we get

$$|\psi\rangle = e^{i\gamma} \left(\cos \frac{\theta}{2} |0\rangle + e^{i\phi} \sin \frac{\theta}{2} |1\rangle \right) \quad (3.2)$$

The overall phase $e^{i\gamma}$ has no effect as it has no *observable effects* and hence can be neglected. So,

$$|\psi\rangle = \cos \frac{\theta}{2} |0\rangle + e^{i\phi} \sin \frac{\theta}{2} |1\rangle \quad (3.3)$$

The numbers θ and ϕ define a point on a unit sphere where θ is the angle from z-axis and ϕ is the angle projection of point makes with the x-axis. It turns out that any operation on a qubit can be represented by a rotation in the Bloch sphere notation.

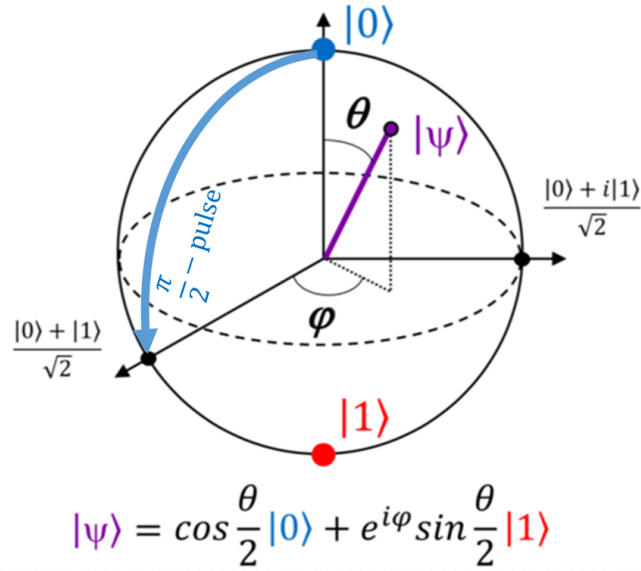


Figure 3.1: Bloch sphere representation of a qubit

3.3 Single Qubit Operations

Any operation on a single qubit is an invertible function $f : |\psi\rangle \rightarrow |\psi\rangle$. In matrix form, it can be written as a 2×2 Unitary matrix U . The criteria that the matrix must be unitary arises from the fact that operations must be reversible and it must preserve the inner product. The following table mentions the most common quantum gates along with their matrix representations.

Operator	Gate(s)	Matrix
Pauli-X (X)	$\text{---} \boxed{\text{X}} \text{---}$ $\text{---} \oplus \text{---}$	$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$
Pauli-Y (Y)	$\text{---} \boxed{\text{Y}} \text{---}$	$\begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$
Pauli-Z (Z)	$\text{---} \boxed{\text{Z}} \text{---}$	$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$
Hadamard (H)	$\text{---} \boxed{\text{H}} \text{---}$	$\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$
Phase (S, P)	$\text{---} \boxed{\text{S}} \text{---}$	$\begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$
$\pi/8$ (T)	$\text{---} \boxed{\text{T}} \text{---}$	$\begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix}$

Figure 3.2: Common single qubit gates

The first three gates (X, Y, Z) are the Pauli spin matrices signifying a rotation of π in the Bloch sphere notation along the respective axes.

The rotations by an angle θ along the \hat{x} , \hat{y} and \hat{z} axes are given by:

$$R_x(\theta) = e^{-i\theta X/2} = \cos \frac{\theta}{2} I - i \sin \frac{\theta}{2} X = \begin{bmatrix} \cos \frac{\theta}{2} & -i \sin \frac{\theta}{2} \\ -i \sin \frac{\theta}{2} & \cos \frac{\theta}{2} \end{bmatrix} \quad (3.4)$$

$$R_y(\theta) = e^{-i\theta Y/2} = \cos \frac{\theta}{2} I - i \sin \frac{\theta}{2} Y = \begin{bmatrix} \cos \frac{\theta}{2} & -\sin \frac{\theta}{2} \\ \sin \frac{\theta}{2} & \cos \frac{\theta}{2} \end{bmatrix} \quad (3.5)$$

$$R_z(\theta) = e^{-i\theta Z/2} = \cos \frac{\theta}{2} I - i \sin \frac{\theta}{2} Z = \begin{bmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{bmatrix} \quad (3.6)$$

Since, every operator can be considered as a rotation in the Bloch sphere notation, there will exist some \hat{n} and θ for all operators U such that U can be seen as a rotation of angle θ about \hat{n} axis.

$$U = R_n(\theta) = e^{-i\theta \hat{n} \cdot \vec{\sigma}/2} = \cos \frac{\theta}{2} I - i \sin \frac{\theta}{2} (n_x X + n_y Y + n_z Z) \quad (3.7)$$

($\vec{\sigma} = X\hat{x} + Y\hat{y} + Z\hat{z}$ is often known as spin matrix vector.)

An important theorem is the $Z - Y$ decomposition of single qubit gates. It says that any unitary operator U can be written in the form

$$U = e^{i\alpha} R_z(\beta) R_y(\gamma) R_z(\delta) \quad (3.8)$$

where $\alpha, \beta, \gamma, \delta$ are some real numbers. An intuitive way to think about this is that we first rotate the axes such that the y-axis aligns with \hat{n} , then perform the rotation and then bring the y-axis back to the correct position by reverse operations. More formally, any unitary operator can be written in the form

$$U = \begin{bmatrix} e^{i(\alpha-\beta/2-\delta/2)} \cos \frac{\gamma}{2} & -e^{i(\alpha-\beta/2+\delta/2)} \sin \frac{\gamma}{2} \\ e^{i(\alpha+\beta/2-\delta/2)} \sin \frac{\gamma}{2} & e^{i(\alpha+\beta/2+\delta/2)} \cos \frac{\gamma}{2} \end{bmatrix} \quad (3.9)$$

and hence the form arises from simple matrix multiplication. In fact, we may replace \hat{z} and \hat{y} by any other non-collinear unit vectors. Another useful form in which the operator can be expressed is

$$U = e^{i\alpha} A X B X C \text{ where } A, B \text{ and } C \text{ are matrices such that } ABC = I \quad (3.10)$$

In terms of the previous representation, we can deduce A, B and C as:

$$\begin{aligned} A &= R_z(\beta) R_y\left(\frac{\gamma}{2}\right) \\ B &= R_y\left(-\frac{\gamma}{2}\right) R_z\left(-\frac{\delta+\beta}{2}\right) \\ C &= R_z\left(\frac{\delta-\beta}{2}\right) \end{aligned} \quad (3.11)$$

3.4 Controlled Operation

The Controlled operations are used to do conditional operations. It takes two qubits, one control qubit and another target qubit, and performs a given operation U on the target qubit if the control qubit is set to $|1\rangle$.

The most famous Controlled operation is the Controlled-NOT(CNOT). It performs the not operation on the target qubit if the control qubit is set. Another way to represent it is CNOT: $|c\rangle|t\rangle \rightarrow |c\rangle|c \oplus t\rangle$.

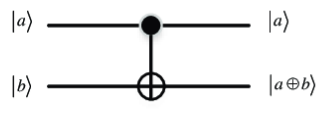
$$CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$


Figure 3.3: Representation and matrix associated with C-NOT gate

Any other controlled operator can be expressed with the help of CNOT gate and single qubit gates using the representation mentioned in Equation 3.10 . The circuit for Controlled- U gate is shown below:

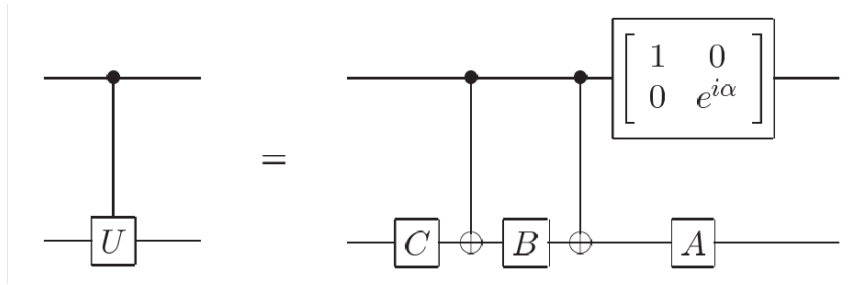


Figure 3.4: Circuit for Controlled- U gate where $U = e^{i\alpha}AXBXC$

The circuit can easily be verified to the controlled operator corresponding to the operator U . When the control qubit(the top one) is set to $|0\rangle$, then no change occurs as $ABC = I$. On the other hand, when the control qubit is $|1\rangle$ then operator $U = e^{i\alpha}AXBXC$ acts on the second qubit(the bottom one).

The controlled operator can be extended to have multiple control and target qubits. Such an operator applies U on the target qubit(s) if all the control qubits are $|1\rangle$. We use the following representation where $|x_1\rangle, |x_2\rangle \dots |x_n\rangle$ are n control qubits and $|\psi\rangle$ is the composite state of the target qubits.

$$C^n(U)|x_1, x_2 \dots x_n\rangle|\psi\rangle = |x_1, x_2 \dots x_n\rangle U^{x_1 x_2 \dots x_n} |\psi\rangle \quad (3.12)$$

Any $C^2(U)$ can be implemented as the following circuit if $U = V^2$ for some unitary matrix V .

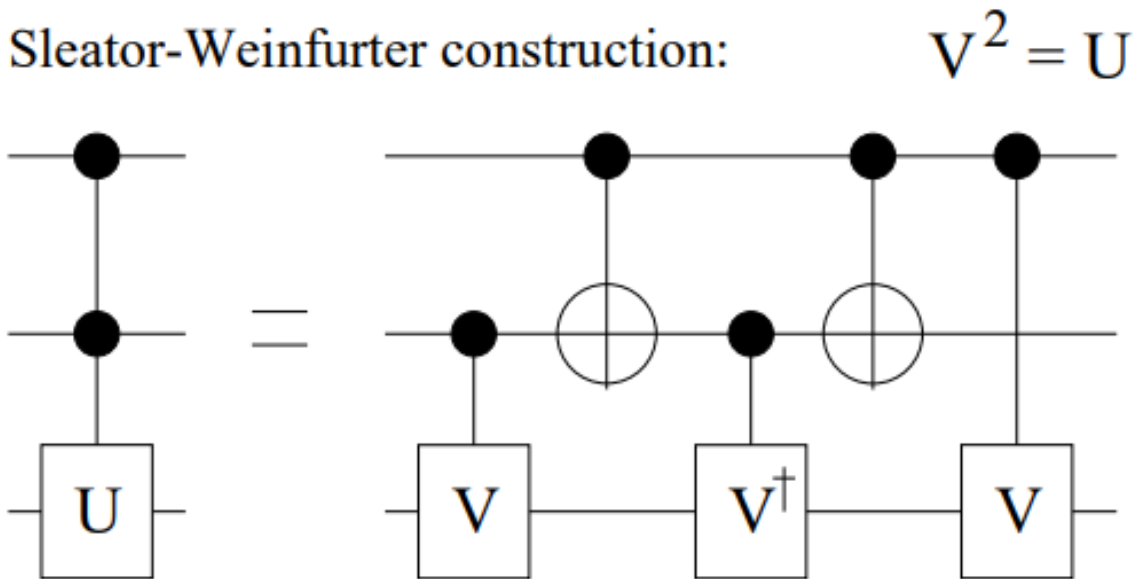


Figure 3.5: Circuit for $C^2(U)$ gate where $U = V^2$

The famous Toffoli gate is $C^2(X)$ and can be implemented as follows:

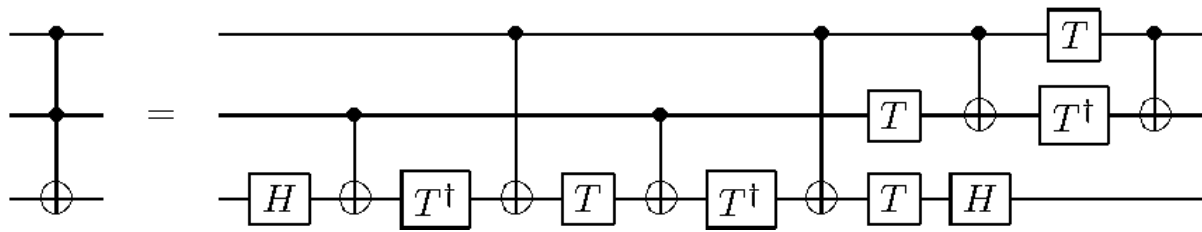


Figure 3.6: Circuit for $C^2(X)$ i.e. Toffoli gate

(Here we have to use both the reductions in the section i.e. C^2 from C^1 and C^1 from CNOT) For general $C^n(U)$ gate we will have to use some extra qubits known as ancilla bits. The following circuit shows how to implement $C^n(U)$:

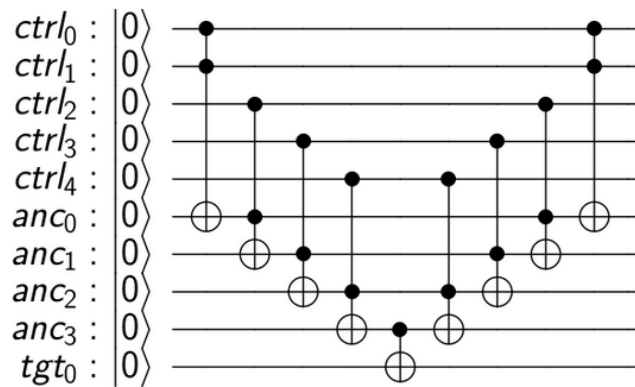


Figure 3.7: Circuit for $C^n(U)$ gate. Here $n=5$

It turns out that any $C^n(U)$ gate can be implemented using $O(n^2)$ Toffoli, CNOT, and single qubit gates. ([This guy](#) even reduced it to $O(n)$ for $U = X$).

3.5 Measurement

The projective measurement is shown in the circuit by the meter symbol. Since any measurement can be expressed in the form of unitary matrix and projective measurement, we will not define any symbol for general measurements.

There are two very important principles about measurement in quantum circuit.// **Principle of deferred measurement:** Measurements can always be moved from an intermediate stage of a quantum circuit to the end of the circuit; if the measurement results are used at any stage of the circuit then the classically controlled operations can be replaced by conditional quantum operations.

Principle of implicit measurement: Without loss of generality, any unterminated quantum wires (qubits which are not measured) at the end of a quantum circuit may be assumed to be measured.

The measurements act as a link between the classical world and the quantum world. The measurements as the only way we can retrieve the information from a quantum system. Hence, measurements are very important part of any system.

3.6 Universal Quantum gates

In the classical domain, it turns out that NAND and NOR are universal gates in the sense that they can be used to form any other gate. But in the quantum case, there is no universal gate but any gate can be approximated to any precision/accuracy using the four gates: Hadamard, phase, $\pi/8$ gate and CNOT.

The proof of universality is broken down in 3 parts:

1. We decompose any multi-qubit gate into single-qubit gates. This means that we can express any unitary operator exactly as a product of unitary operators that each acts non-trivially only on a subspace spanned by two computation basis states.
2. Then, we prove that any two qubit gate can be expressed exactly in terms of single qubit gates and CNOT.
3. Then we approximate the single qubit gates by using the phase, hadamard and $\pi/8$ gates.

3.6.1 Two-level gates are Universal

This is the first step in proving the universality of Hadamard, $\pi/8$, phase and CNOT gates.

We will show that a unitary matrix acting on d -dimensional Hilbert space may be decomposed into a product of two-level unitary matrices.

The basic idea is we recursively eliminate one element from the first column of the matrix until only one element is left by multiplying with some two-level unitary matrices. Then, we do the same thing for all the other columns leaving I . Then, the matrix can be expressed as the product of conjugate of the matrices multiplied. These all will be two-level unitary matrices and hence our decomposition will be complete.

Let us consider the case for 3×3 matrix U . Let

$$U = \begin{bmatrix} a & d & g \\ b & e & h \\ c & f & j \end{bmatrix} \quad (3.13)$$

If $b \neq 0$, then multiply by U_1 given by

$$U_1 = \begin{bmatrix} \frac{a^*}{\sqrt{|a|^2+|b|^2}} & \frac{b^*}{\sqrt{|a|^2+|b|^2}} & 0 \\ \frac{b}{\sqrt{|a|^2+|b|^2}} & \frac{-a}{\sqrt{|a|^2+|b|^2}} & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (3.14)$$

It is easy to see that this is a two-level unitary matrix.

$$U_1 U = \begin{bmatrix} a' & d' & g' \\ 0 & e' & h' \\ c' & f' & j' \end{bmatrix} \quad (3.15)$$

On, similar lines, now multiply by U_2 , given by

$$\begin{aligned} \text{if } b = 0, \text{ then } U_2 &= \begin{bmatrix} a'^* & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ \text{else } U_2 &= \begin{bmatrix} \frac{a'^*}{\sqrt{|a'|^2+|c'|^2}} & 0 & \frac{c'^*}{\sqrt{|a'|^2+|c'|^2}} \\ 0 & 1 & 0 \\ \frac{c'}{\sqrt{|a'|^2+|c'|^2}} & 0 & \frac{-a'}{\sqrt{|a'|^2+|c'|^2}} \end{bmatrix} \end{aligned} \quad (3.16)$$

This reduces U to

$$\begin{aligned} U_2 U_1 U &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & e'' & h'' \\ 0 & f'' & j'' \end{bmatrix} \\ U &= U_1^\dagger U_2^\dagger \begin{bmatrix} 1 & 0 & 0 \\ 0 & e'' & h'' \\ 0 & f'' & j'' \end{bmatrix} \end{aligned} \quad (3.17)$$

Clearly, all the matrices on the *RHS* are two-level unitary matrices and hence we have successfully decomposed U in terms of three unitary matrices.

Extending the idea above, we see that to decompose d dimensional unitary matrix, we would need $d(d-1)/2$ two-level matrices or $O(d^2)$ two level matrices.

3.6.2 Single Qubit gates and CNOT are universal

This is easy to show. The main thing lies in how to transform the given two-level unitary operator into a matrix that has effect only on one qubit. After this, we can just use $C^n(U)$ from the other qubits to this qubit and perform the operation. Then, we can undo the transformation to get the original states of the other qubits.

To perform this transformation, we use *Gray codes*. A Gray code connecting s and t is a sequence of binary numbers, starting with s and concluding with t , such that adjacent members of the list differ in exactly one bit. So, we form the Gray code from one of the states on which the given matrix to other. Then, we transform the first state into the second-last member of the list(which is differing from the other state at only one place). These operations may be accomplished using Controlled NOT from the identical qubits to the differing qubit. Since we are using conditional NOTs, this does not change other basis states. Then apply the given controlled two-level matrix(in reduced form i.e. the 2×2 matrix that is non-trivial) on the qubit which is different conditioned on the other identical qubits. Then, transform the state back into its original states by again applying the same controlled-NOTs in reverse order.

This transformation, operation and then coming back takes $O(n)$ multi-qubit CNOTs along with the Controlled- U operation(U is the reduced two-level matrix). Here, n is the number of qubits. Since, each CNOT takes $O(n)$ single qubit CNOT gates, it turns out that we would need $O(n^2)$ gates in total to realise the given two-level operation. Any n qubit unitary operator is 2^n dimensional and hence takes $O(4^n)$ two-level unitary operators. So in total, to perform a n -qubit operation, we need $O(4^n n^2)$ single qubit gates and CNOTs. This is exponential in the number of qubits!!! And it turns out that there DO exist some unitary operators which need this much gates.

3.6.3 Approximating Single-Qubit Gates

For approximating, we must first have measure to quantify the approximation i.e. a value which can tell how good the approximation is. In this case, we use the following function as the error function:

$$E(U, V) = \max_{|\psi\rangle} \|(U - V)|\psi\rangle\| \quad (3.18)$$

Here, U is the target or the actual operator and V is the operator used to approximate it. The error function is quite intuitive and tells us how much correct the approximation is in terms of

the maximum error caused by the approximation. An important property of this error function is

$$E(U_m U_{m-1} \dots U_2 U_1, V_m V_{m-1} \dots V_2 V_1) \leq \sum_{j=1}^m E(U_j, V_j) \quad (3.19)$$

This says that the error in approximating multiple operators is at most the sum of individual errors. So, we want to approximate m unitary operators with a total error of Δ , then it is enough to approximate the individual operators upto an error of $\frac{\Delta}{m}$.

Now, we move on to approximate any single qubit unitary operator using Hadamard, phase and $\pi/8$ gates.

In this section, the $\pi/8$ gate is denoted by T . The T performs a rotation of $\pi/8$ around the \hat{z} axis. Similarly, the operator HTH performs a rotation of $\pi/8$ around the \hat{x} axis. From this we get

$$\begin{aligned} HTH &= e^{-\frac{i\pi X}{8}} \cdot e^{-\frac{i\pi Z}{8}} \\ &= \left[\cos \frac{\pi}{8} I - i \sin \frac{\pi}{8} X \right] \left[\cos \frac{\pi}{8} I - i \sin \frac{\pi}{8} Z \right] \\ &= \cos^2 \frac{\pi}{8} I - i \left[\cos \frac{\pi}{8} (X + Z) + \sin \frac{\pi}{8} Y \right] \sin \frac{\pi}{8} \end{aligned} \quad (3.20)$$

This is a rotation in the bloch sphere notation about the axis given by $\bar{n} = (\cos \frac{\pi}{8}, \sin \frac{\pi}{8}, \cos \frac{\pi}{8})$ through an angle of θ defined by $\cos \theta/2 = \cos^2 \frac{\pi}{8}$. This θ is an irrational multiple of 2π and so any angle can be approximated to any precision using this θ . Also note that $HR_{\bar{n}}(\alpha)H = R_{\bar{m}}(\alpha)$ where \bar{m} is some vector not aligned with \bar{n} . Since $R_{\bar{n}}(\alpha)$ can be approximated to any precision/accuracy using the θ defined before, $R_{\bar{m}}(\alpha)$ can be approximated to any precision/accuracy. So, we may approximate any unitary matrix using $R_{\bar{m}}(\alpha)$ and $R_{\bar{n}}(\alpha)$ by equation 3.8 for general non-collinear vectors. So, for any arbitrary U we have

$$U = R_{\bar{n}}^a R_{\bar{m}}^b R_{\bar{n}}^c \quad (3.21)$$

for some whole numbers a, b and c . If we have to approximate U to an accuracy of ϵ , then we have to approximate each operator by $\frac{\epsilon}{a+b+c}$.

So, it turns out that we can approximate any unitary operator using CNOT, Hadamard and $\pi/8$ gate. In addition to this, Phase gate is used to increase the fault tolerance of the circuit. So, these four gates form a complete set of universal gates.

3.7 Solovay-Kitaev Theorem

In the previous section, we approximated any single qubit gates using the Hadamard, Phase and $\pi/8$ gates. But, how many of these universal gates are needed to simulate a gate? This question is answered by the **Solovay-Kitaev Theorem**. According to this theorem, any gate can be simulated upto a tolerance of ϵ using $O(\log^c(\frac{1}{\epsilon}))$ universal gates where c is constant between 1 and 2.

3.7.1 Proof of Solovay Kitaev theorem

SOLOVAY KITAEV THEOREM: Let G be a finite set of elements in $SU(2)$ containing its own inverses, such that $\langle G \rangle$ is dense in $SU(2)$. Let $\epsilon > 0$ be given. Then G_l is an ϵ -net in $SU(2)$ for $l = O(\log^c(1/\epsilon))$, where $c \simeq 4$. (The actual value of c is less but it is easier to show for 4).

Notations used

1. **$SU(2)$:** The set of all single qubit unitary matrices with determinant equal to one. Any single qubit unitary operator can be written as a product of elements of $SU(2)$ upto an unimportant global factor.
2. G : Finite set of elements of $SU(2)$ which is used to simulate all other operators. In our case it is H, S, T gates along with their inverses with some global factor s.t. the determinant is 1.
3. G_l : Set of all words of length l formed by elements of G .
4. $\langle G \rangle$: Union of G_l for all l .
5. **.Distance Function:** We need a distance function to quantify the approximation. We use trace distance in this case,

$$D(U, V) = \text{tr}|U - V| \text{ where } |U| \equiv \sqrt{U^\dagger U} \quad (3.22)$$

6. **Dense Subset and ϵ -net:** A subset is said to be dense if for all $\epsilon > 0$ and for all elements of the set, there exists an element in the subset such that their distance is less than ϵ . A subset S is said to be ϵ -net of another subset W if all elements of W lie within an ϵ distance from some element of S .

The proof involves repetitive use of the following lemma:

Lemma : Let G be a finite set of elements in $SU(2)$ containing its own inverses, and such that $\langle G \rangle$ is dense in $SU(2)$. There exists a universal constant ϵ_0 independent of G , such that for any $\epsilon \leq \epsilon_0$, if G_l is an ϵ^2 -net for S_ϵ , then G_{5l} is a $C\epsilon^3$ -net for $S_{\sqrt{C}\epsilon^{3/2}}$ for some constant C . Here S_ϵ denotes an ϵ neighbourhood of I .

Since $\langle G \rangle$ is dense in $SU(2)$ we can find l_0 such that G_{l_0} is an ϵ_0^2 -net for $SU(2)$ and hence for S_{ϵ_0} . Applying the above lemma, we get that for $l = 5l_0$, G_l is $C\epsilon_0^3$ -net for $S_{\sqrt{C}\epsilon_0^{3/2}}$. Repetitively applying the above lemma, we get for $l = 5^k l_0$, G_l is $\epsilon(k)^2$ -net for $S_{\epsilon(k)}$ where

$$\epsilon(k) = \frac{(C\epsilon_0)^{(3/2)^k}}{C} \quad (3.23)$$

We may assume $C\epsilon_0 < 1$ and so $\epsilon(k)$ decreases with k . So, we may get as close to I with as much accuracy as needed.

Now let $U \in SU(2)$ be the operator to be approximated. Let $U_0 \in G_{l_0}$ be $\epsilon(0)^2$ -approximation of U . Let $V_0 = UU_0^\dagger$. Then

$$D(V_0, I) = \text{tr}|V_0 - I| = \text{tr}|U - U_0| < \epsilon(0)^2 < \epsilon(1) \quad (3.24)$$

So, V_0 lies in $S_{\epsilon(1)}$ and hence we have $\epsilon(1)^2$ -approximation of V_0 in G_{5l_0} say U_1 . Continuing in the same way as before, we define $V_1 = V_0U_1^\dagger = UU_0^\dagger U_1^\dagger$ and find that V_1 lies in $S_{\epsilon(2)}$. By repetitively doing this, we can approximate U to $\epsilon(k)^2$ for any k . For this number of gates required are $l_0 + 5l_0 + \dots + 5^k l_0 \simeq \frac{5}{4}5^k l_0$. If we want an accuracy of ϵ , then we would want k such that $\epsilon(k)^2 < \epsilon$. Substituting $\epsilon(k)$, we get

$$\left(\frac{3}{2}\right)^k < \frac{\log(1/C^2\epsilon)}{2\log(1/C\epsilon_0)} \quad (3.25)$$

So, number of gates required to approximate within ϵ in $O(\log^c(1/\epsilon))$ where $c = \frac{\log 5}{\log 3/2} \approx 4$. Hence, the Solovay-Kitaev Theorem is proved.

Part II

Algorithms

Chapter 4

Deutsch-Josza Algorithm

4.1 Problem

Given a hidden boolean function f acting on a string of bits, find out whether the function f is a constant or balanced function. It is guaranteed that the function would definitely be one of these.

$$\begin{aligned} \text{Constant: } f(\{x_0, x_1, x_2 \dots\}) &= 0 \text{ or } 1 \forall x_0, x_1, x_2 \dots \in \{0, 1\}^{\otimes n} \\ \text{Balanced: } f(\{x_0, x_1, x_2 \dots\}) &= 0 \text{ half of the times and } 1 \text{ otherwise} \end{aligned} \quad (4.1)$$

4.2 Classical Solution

In the worst case for the classical domain, we would have to check for half of the numbers plus one to determine if the function is balanced or not. So, if there are n bits, a classical computer would take $2^{n-1} + 1$ computations.

For this worst case to happen, we must get the same value for all the 2^{n-1} numbers we checked first. The probability of this happening is very low for the balanced case and often we can find that the function is balanced much earlier by using a randomized selection for numbers. but in the case of constant function, we would have to check for this many times to be 100% sure.

4.3 Quantum Solution

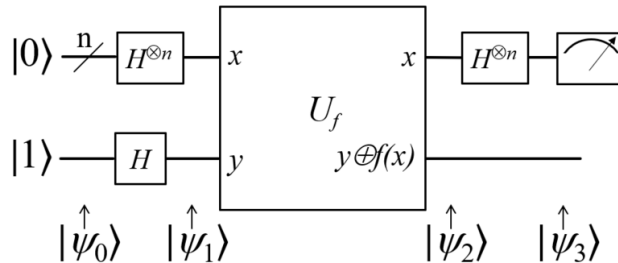


Figure 4.1: Circuit for Deutsch-Josza Algorithm

The above circuit can be used to solve the Deutsch-Josza Problem in one call only!!! The function U_f takes $|x\rangle|y\rangle$ and outputs $|x\rangle|y \oplus f(x)\rangle$. it is often referred to as *Quantum Oracle* because we just assume we have such a gate without actually knowing f . Let us see what happens in this circuit:

1. We take the state $|\psi_0\rangle = |0\rangle^{\otimes n}|1\rangle$ where the first n bits act as the input register and last bit is used to store the result.
2. Apply Hadamard gate on all the bits. The state now becomes

$$|\psi_1\rangle = \frac{1}{\sqrt{2^{n+1}}} \sum_{x=0}^{2^n-1} |x\rangle(|0\rangle - |1\rangle) \quad (4.2)$$

Here, the first n bits are used to represent the number x .

3. Next, we apply the oracle on $|\psi_1\rangle$ to get

$$\begin{aligned} |\psi_2\rangle &= \frac{1}{\sqrt{2^{n+1}}} \sum_{x=0}^{2^n-1} |x\rangle(|0 \oplus f(x)\rangle - |1 \oplus f(x)\rangle) \\ &= \frac{1}{\sqrt{2^{n+1}}} \sum_{x=0}^{2^n-1} (-1)^{f(x)} |x\rangle(|0\rangle - |1\rangle) \end{aligned} \quad (4.3)$$

Notice that the last qubit value remains unchanged and the application of the quantum oracle actually changes the phase of the various values of the first register(the $|x\rangle$) depending on the value of $f(x)$.

4. Now, again apply the Hadamard gate but now only on the first register. Since there is no operation on the last qubit, it remains unchanged. So, we may focus on the values of the first register only.

$$\begin{aligned} |\psi_3\rangle_{\text{register1}} &= \frac{1}{2^n} \sum_{x=0}^{2^n-1} (-1)^{f(x)} \left(\sum_{z=0}^{2^n-1} (-1)^{x.z} |z\rangle \right) \\ &= \frac{1}{2^n} \sum_{z=0}^{2^n-1} \left(\sum_{x=0}^{2^n-1} (-1)^{x.z+f(x)} \right) |z\rangle \end{aligned} \quad (4.4)$$

Notice that here, we have used the general form of Hadamard Gate :

$$H^{\otimes n}|x\rangle = \frac{1}{\sqrt{2^n}} \left(\sum_{z=0}^{2^n-1} (-1)^{x.z} |z\rangle \right) \quad (4.5)$$

where $x.z = x_0z_0 \oplus x_1z_1 \oplus \dots \oplus x_nz_n$ is the XOR of bit-wise product.(Verify this!!)

5. Lastly, we measure the value of first register. This will result in the collapsing of the first register in one of the states $|z\rangle$.
 - If f is a constant function, the coefficient of the state $|0\rangle^{\otimes n}$ is 1 or -1 depending on the value of $f(x)$. This means that the probability of measuring this state is 1. So, if the function is constant, we would observe the $|0\rangle^{\otimes n}$ state.
 - If f is a balanced function, the coefficient of the state $|0\rangle^{\otimes n}$ is 0 as $\sum (-1)^{x.0+f(x)} = \sum (-1)^{f(x)} = 0$. So, there is zero probability of observing this state.

Therefore by looking at the value of the first register, we can tell whether the function is constant or balanced.

This problem shows that quantum computer may offer an exponential speed over the classical computer. This is the case with other algorithms too.

Chapter 5

Simon's Algorithm

5.1 Problem

In the Simon's Problem, we have a function f which is one-one or two-one, i.e. the values for each input may be unique(one-one) or two inputs may the same output(two-one). Further, if the function is two-one, we have a hidden bit-string s such that

$$\text{if } f(x_1) = f(x_2) \text{ then } x_1 \oplus x_2 = s \quad (5.1)$$

our objective is to find whether f is one-one or in the other case, find the bit-string s .

5.2 Classical Soution

In the classical regime, the worst case scenario takes $2^{n-1} + 1$ calls to the function f where n is the number of bits required to represent the input of f . In the worst case, all the numbers that we sample returns different values of $f(x)$. This can happen for at max 2^{n-1} times. hence, we would require one more call to get the final result.

5.3 Quantum Solution

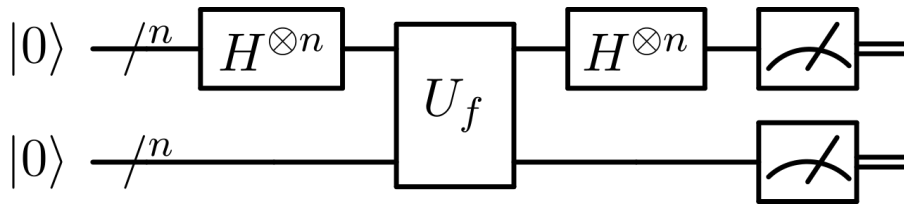


Figure 5.1: Circuit for Simon's Algorithm

The above circuit shows the circuit used to solve the Simon's Problem. The circuit is similar to the Deutsch-Josza one. Here again, U_f is the quantum oracle for the function f . Let us see what each step does.

1. The initial state consist of two registers consisting of n qubits. Both are initialised to $|0\rangle^{\otimes n}$. So, $|\psi_0\rangle = |0\rangle^{\otimes n}|0\rangle^{\otimes n}$
2. First we apply Hadamard gate to the first register. the vaue of the qubits changes to:

$$|\psi_1\rangle = \frac{1}{\sqrt{2^n}} \left(\sum_{x=0}^{2^n-1} |x\rangle \right) |0\rangle^{\otimes n} \quad (5.2)$$

3. Then we apply the quantum oracle changing the second register to $f(x)$. The new state is

$$|\psi_2\rangle = \frac{1}{\sqrt{2^n}} \left(\sum_{x=0}^{2^n-1} |x\rangle |f(x)\rangle \right) \quad (5.3)$$

4. Applying Hadamard again, and using the general transformation for $H^{\otimes n}$,

$$|\psi_3\rangle = \frac{1}{2^n} \sum_{x=0}^{2^n-1} \left(\sum_{z=0}^{2^n-1} (-1)^{x \cdot z} |z\rangle \right) |f(x)\rangle \quad (5.4)$$

5. Measure the two registers. The second register is measured first. When the second register is measured, the value of first register collapses to the values which have f as measured with Hadamard applied on them. So,

$$\text{For one-one, } |\psi_4\rangle = \frac{1}{\sqrt{2^n}} \sum_{z=0}^{2^n-1} (-1)^{x \cdot z} |z\rangle \quad (5.5)$$

$$\begin{aligned} \text{For two-one, } |\psi_4\rangle &= \frac{1}{\sqrt{2^{n+1}}} \left(\sum_{z=0}^{2^n-1} (-1)^{x \cdot z} |z\rangle + \sum_{z=0}^{2^n-1} (-1)^{y \cdot z} |z\rangle \right) \\ &= \frac{1}{\sqrt{2^{n+1}}} \sum_{z=0}^{2^n-1} ((-1)^{x \cdot z} + (-1)^{y \cdot z}) |z\rangle \end{aligned} \quad (5.6)$$

Note that by the given condition defining s , we have $y = x \oplus s$.

6. now measure the first register. If the value is $|0\rangle^{\otimes n}$, this means that $(-1)^{x \cdot z} = (-1)^{y \cdot z}$. If this equality does not hold, we get some other value. Notice the condition.

$$\begin{aligned} (-1)^{x \cdot z} &= (-1)^{y \cdot z} \\ \Rightarrow x \cdot z &= y \cdot z \pmod{2} \\ \Rightarrow x \cdot z &= (x \oplus s) \cdot z \pmod{2} \\ \Rightarrow x \cdot z &= x \cdot z \oplus s \cdot z \pmod{2} \\ \Rightarrow s \cdot z &= 0 \pmod{2} \end{aligned} \quad (5.7)$$

If the above equation does not hold, then $s \cdot z = 1 \pmod{2}$.

If we have to find s , we have to basically find the value of each of the bits. For this, we will need n equations as s is of n bits. So, repeating the above process till we get n different z , we will get n different equations and hence the value of s can be determined.

The above algorithm needs about $O(p(n))$ runs of the above circuit, where $p(\cdot)$ is a polynomial function. Again, using quantum computer leads to an exponential speedup.

Chapter 6

Quantum Fourier Transform

6.1 Fourier Transform

The Fourier Transform of a discrete set of numbers x_0, x_1, \dots, x_{N-1} is defined as the set of numbers y_0, y_1, \dots, y_{N-1} where :

$$y_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j e^{2\pi i j k / N} \quad (6.1)$$

where $i = \sqrt{-1}$. Like any other transformation, Fourier transform also has an inverse:

$$FT^{-1}(\{y_0, y_1, \dots, y_{N-1}\}) = \{x_0, x_1, \dots, x_{N-1}\} \text{ where } x_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} y_j e^{-2\pi i j k / N} \quad (6.2)$$

In case of quantum systems, Fourier transform is defined to be the linear operator:

$$\sum_{j=0}^{N-1} x_j |j\rangle \rightarrow \sum_{k=0}^{N-1} y_k |k\rangle \quad (6.3)$$

where the sets, $|j\rangle$ and $|k\rangle$, are the basis states and $\{y_0, y_1, \dots, y_{N-1}\}$ are the classic Fourier transform of $\{x_0, x_1, \dots, x_{N-1}\}$.

Fourier Transform is useful in a variety of fields like signal processing, image processing etc. and hence computing the Fourier Transform is essential.

6.2 Classical Solution

In case of classical computers, the best known algorithm for computing Fourier Transform is the Fast Fourier Transform(FFT). The time-complexity of this algorithm is $O(N \log N)$ where N are the number of elements in the sequence.

FFT is the set of algorithms which uses the divide and conquer strategy to compute the Discrete Fourier Transform. The most common of them is the [Cooley-Tukey FFT algorithm](#). It divides the set of numbers into two sets based on whether they are even or odd. Then, it computes the Fourier Transform E_k and O_k of these smaller sets. Then, the fourier transform of the whole set is given by:

$$\begin{aligned} Y_k &= E_k + e^{-\frac{2\pi i}{N}k} O_k \\ Y_{k+\frac{N}{2}} &= E_k - e^{-\frac{2\pi i}{N}k} O_k \end{aligned} \quad (6.4)$$

The algorithm can be used to compute the FT on the series of even and odd sets i.e. E_k and O_k . Note that the algorithm divides the problem of FT of a series into two problems of computing the FT on sub-series and hence it is a divide and conquer algorithm. The combining step takes $O(n)$ time and hence the Time complexity equation becomes:

$$T(N) = 2T\left(\frac{N}{2}\right) + O(N) \quad (6.5)$$

By using the Master's Theorem, we get that $T(N) = O(N \log N)$.

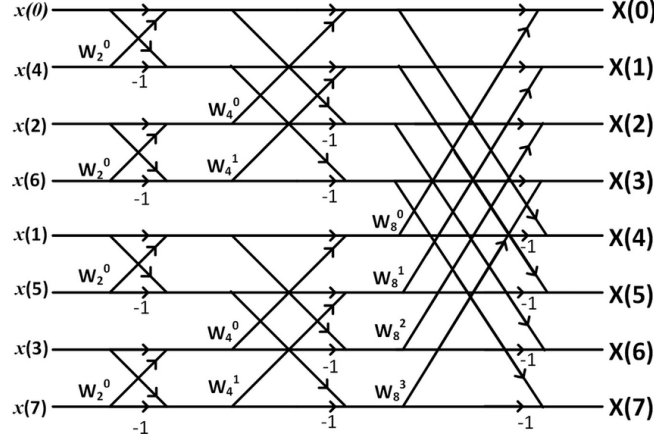


Figure 6.1: Butterfly diagram for the Cooley-Tukey FFT algorithm

6.3 Quantum Solution

In the case of Quantum Computing, the complexity to calculate the Quantum FT is $O(n^2)$ where $N = 2^n$, an exponential speed-up as compared to the classical case!!

To understand the algorithm used for the QFT computation, we must first re-write the expression for QFT.

$$\begin{aligned}
 QFT_N|x\rangle &= \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} e^{\frac{2\pi i y x}{N}} |y\rangle \\
 &= \frac{1}{\sqrt{2^n}} \sum_{y=0}^{N-1} e^{\frac{2\pi i y x}{2^n}} |y\rangle \\
 &= \frac{1}{\sqrt{2^n}} \sum_{y=0}^{N-1} e^{2\pi i x \frac{\sum_{k=1}^n y_k}{2^k}} |y_1 y_2 \dots y_n\rangle \text{ where } (y_1 y_2 \dots y_n) \text{ is the binary representation of } y \\
 &= \frac{1}{\sqrt{2^n}} \sum_{y=0}^{N-1} \prod_{k=1}^n e^{2\pi i x \frac{y_k}{2^k}} |y_1 y_2 \dots y_n\rangle \\
 &= \frac{1}{\sqrt{2^n}} \bigotimes_{k=1}^n \left(|0\rangle + e^{\frac{2\pi i x}{2^k}} |1\rangle \right) \\
 &= \frac{1}{\sqrt{2^n}} \bigotimes_{k=1}^n \left(|0\rangle + e^{2\pi i (0.x_{n-k+1} x_{n-k+2} \dots x_n)} |1\rangle \right)
 \end{aligned}$$

where $(x_1 x_2 \dots x_n)$ is the binary representation of x

(6.6)

In the last step, we use the periodicity of $e^{2\pi i x}$, $e^{2\pi i x} = e^{2\pi i(x+1)}$. Also, the number $0.x_{n-k+1}x_{n-k}\dots x_n$ is a fraction in the binary system i.e. $0.x_{n-k+1}x_{n-k+2}\dots x_n = \frac{x_{n-k+1}}{2} + \frac{x_{n-k}}{2^2} \dots + \frac{x_n}{2^k}$. The formula at the last step is often called the *product representation* of QFT.

The *product representation* allows us to make a circuit to calculate the QFT. Note that using the *product representation*, we can say that the QFT takes n qubits and transform the k^{th} qubit to $(|0\rangle + e^{2\pi i(0.x_{n-k+1}x_{n-k+2}\dots x_n)}|1\rangle)$. This shows that the k^{th} qubit state depends on the last k qubits. This is a problem as if $k > n/2$, then the value of the qubits that is to be used are already changed and hence we don't have access to x_j for those values. To tackle this, we reverse the sequence of qubits. So now, the transformation becomes $x_k \rightarrow (|0\rangle + e^{2\pi i(0.x_k x_{k+1}\dots x_n)}|1\rangle)$. This can be done as now, the state of k^{th} qubit depends on the value after that. So, the QFT is calculated in two steps.

1. Perform the transformation $x_k \rightarrow (|0\rangle + e^{2\pi i(0.x_k x_{k+1}\dots x_n)}|1\rangle)$ for each qubit.
2. Swap the k^{th} qubit with $(n - k + 1)^{th}$ qubit for each $k < n/2$. This step is easy and can be accomplished by using three *CNOT* gates per swap. So in all, this step requires $O(n)$ *CNOT* gates.

For the 1^{st} step, we require an additional set of gates R_k ($k \in 1, 2, \dots, n$):

$$R_k = \begin{bmatrix} 1 & 0 \\ 0 & e^{2\pi i/2^k} \end{bmatrix} \quad (6.7)$$

These gates can all be formed using the universal set of gates. Also, note that $R_1 = H$.

So, let us say we have to perform the 1^{st} step on the k^{th} qubit. By using the expansion of $0.x_k x_{k+1}\dots x_n$, we see that the transformation is actually *Controlled* $- R_i$ from the $(k + i - 1)^{th}$ qubit for $i \in 1, 2, \dots, (n - k + 1)$. So, by repeatedly performing this operation for all the qubits, we will accomplish the first step. The circuit for the 1^{st} step is shown below:

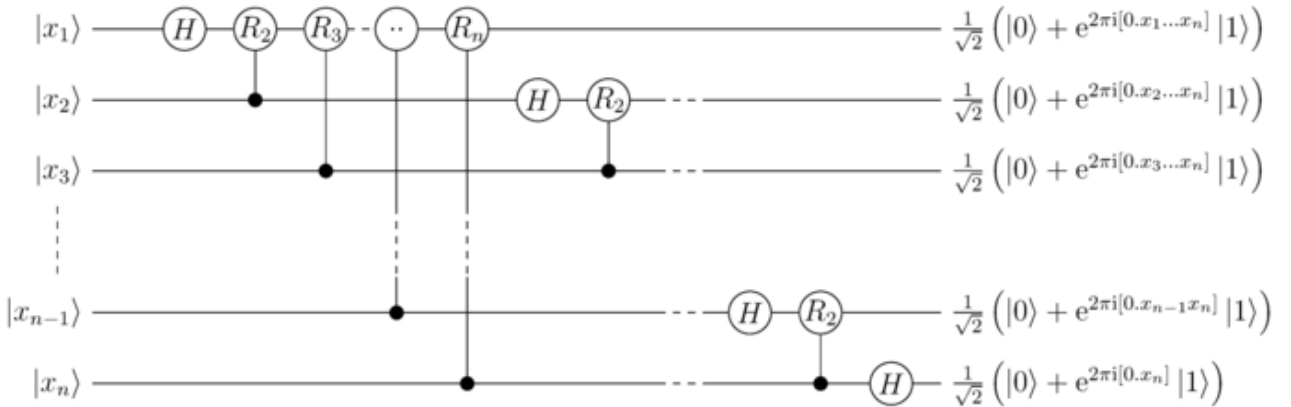


Figure 6.2: Circuit for the first step of QFT

The number of gates required to perform this operation is $O(n^2)$ as the k^{th} qubit requires $(n - k + 1)$ gates. So in all it requires $O(n^2)$ gates to compute the whole QFT. The complete QFT circuit for $n = 3$ is shown on the next page. Note that it requires only 7 (or 9 if we consider the *SWAP* gate as 3 *CNOT* gates) which is in $O(n^2)$.

The inverse of *QFT* i.e. QFT^\dagger can be calculated in a similar way as the only difference in the Fourier and inverse Fourier transform of a series is a negative sign in the exponent. So, by just modifying the R_k gates to have a negative in the exponent, we can compute the inverse transform by using the same circuit.

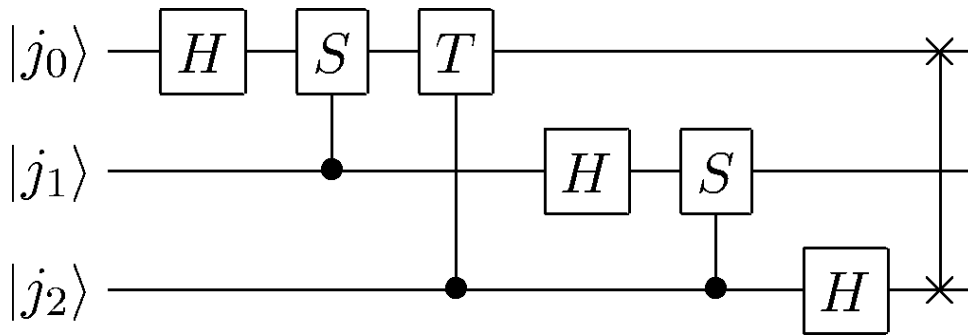


Figure 6.3: Circuit for calculating QFT for $n=3$

It turns out that the exponential speed-up for calculating the Fourier Transform has a lot of uses. This allows us to do all the tasks that require the Fourier Transform in exponentially less time. Most of the algorithms(except for the Quantum Search), including the Shor's Algorithm used to compute the factorisation, that are formed for quantum computing uses this transformation as an essential part.

Chapter 7

Applications of Quantum Fourier Transform

7.1 Quantum Phase Estimation

7.1.1 Problem

Given a unitary operator U and one of its eigen vector $|u\rangle$, determine the phase of the eigen value corresponding to $|u\rangle$. The phase of a unit modulus complex number $e^{2\pi i\phi}$ is ϕ . Note that the phase ϕ will be a fraction between 0 and 1 due to periodicity.

7.1.2 Solution

Since the quantum phase estimation is a general problem in the sense that the unitary operator U is not defined, we would assume that we have an oracle which can perform the unitary transformation corresponding to U . Further, by using multiple oracles like this, we may form the *Controlled- U^{2^j}* gate for any non-negative integer j . How to make this oracle will depend on the specific application and is not a problem to be considered in the quantum phase estimation.// Having this oracle and the algorithm to find QFT^\dagger with us, we can solve this problem easily using the following circuit.

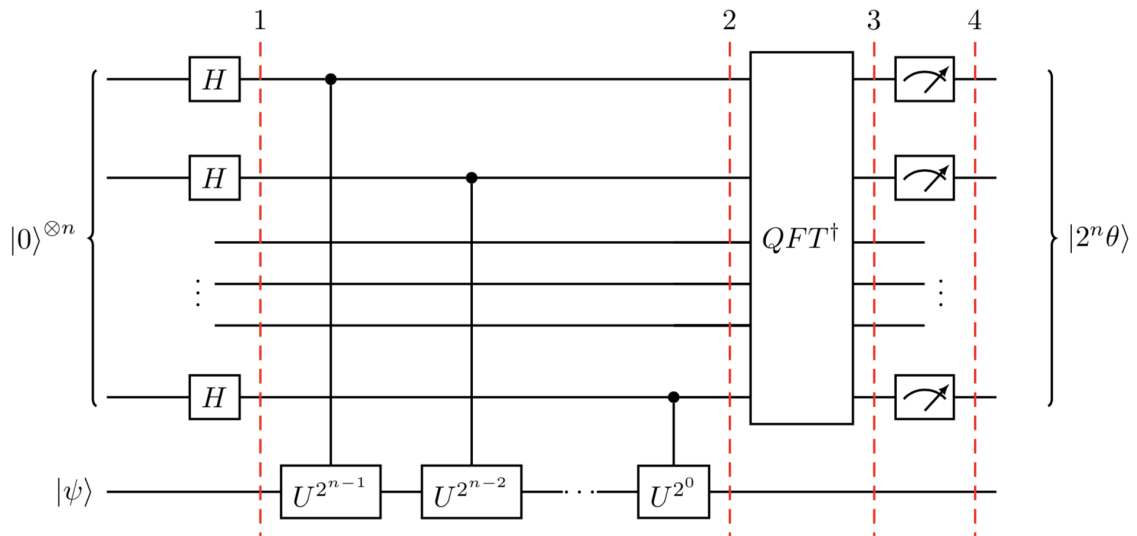


Figure 7.1: Circuit for quantum phase estimation

We use two registers, one is the output register and the other one contains the eigen vector $|\psi\rangle$. The size of the output register is determined by how much precision and accuracy is needed in the answer. If we want to estimate the phase to an accuracy of 2^{-x} and want the probability of success to be ϵ , then the number of qubits in the output register n , will be

$$n = x + \left\lceil \log \left(2 + \frac{1}{2(1-\epsilon)} \right) \right\rceil \quad (7.1)$$

The second term is used to ensure that the result we get is correct with probability 1. The circuit is divided into 4 parts:

1. **Preparing the states:** The initial state taken is $|0\rangle^{\otimes n}|\psi\rangle$ and Hadamard gate is applied on the first register. This is done to get all the possible values of the first register in a superposed state. The transformation is

$$|0\rangle^{\otimes n}|\psi\rangle \rightarrow \frac{1}{2^{n/2}} \sum_{x=0}^{2^n-1} |x\rangle|\psi\rangle = \bigotimes_n \left(\frac{|0\rangle + |1\rangle}{\sqrt{2}} \right) |\psi\rangle \quad (7.2)$$

2. **Applying the operator:** This is the most important part of the phase estimation algorithm. *Controlled* $- U^{2^{n-j}}$ is applied from the j^{th} qubit of the first register to the second register. Note that the value of the qubits in the first register is $(|0\rangle + |1\rangle)/\sqrt{2}$. So, the overall operation is:

$$\bigotimes_n \left(\frac{|0\rangle + |1\rangle}{\sqrt{2}} \right) |\psi\rangle \rightarrow \bigotimes_{j=1}^n \left(\frac{|0\rangle + e^{2\pi i \phi 2^{n-j}} |1\rangle}{\sqrt{2}} \right) |\psi\rangle \quad (7.3)$$

Note the value of exponent $2\pi i \phi 2^{n-j}$. If $0.\phi_1\phi_2\dots\phi_n$ is the binary representation of the phase ϕ , then $2^{n-j}\phi$ becomes $\phi_1\phi_2\dots\phi_{n-j}.\phi_{n-j+1}\dots\phi_n\dots$. But due to the periodic property, it is equivalent to $0.\phi_{n-j+1}\dots\phi_n\dots$. So the state becomes:

$$\bigotimes_{j=1}^n \left(\frac{|0\rangle + e^{2\pi i 0.\phi_{n-j+1}\dots\phi_n} |1\rangle}{\sqrt{2}} \right) |\psi\rangle \quad (7.4)$$

This is the *product representation* of the Fourier Transform of $|\tilde{\phi}\rangle$. Note that we have denoted it as $\tilde{\phi}$ and not ϕ . This is because the value may not be represented in n qubits and hence it will be an approximation.

3. **Inverse Fourier Transform:** This is quite obvious step. So after this step we will get $|\tilde{\phi}\rangle$. Note that the Inverse Fourier transform will not be exact (the binary representation continues after n terms). This leads to some extra terms other than the approximation $|\tilde{\phi}\rangle$ and hence the probability of getting the correct phase approximation may not be 1. Even with this fact, by repeatedly doing the experiment and using some extra qubits (mentioned in the formula for the size of output register), this probability can be made very close to 1.
4. **Measure:** This step is not necessary if the phase is to be used in some further process. But in case of just quantum phase estimation, we do want to measure the value of the phase. Also, measurement is helpful in removing the extra "noise" terms that occur due to approximate Inverse Fourier Transform.

One drawback of this algorithm is that we must have the eigen vector $|\psi\rangle$. This may be possible in some cases where the eigenvector is obtained as a result of some other operation. But even in case of some other vector, the outcome is not so bad. Let us say that the input $|\psi\rangle = \sum c_j |u_j\rangle$ where $|u_j\rangle$ are the eigen bases. Then this operation will lead to $\sum c_j |\tilde{\phi}_j\rangle |u_j\rangle$ with some noise terms. If we measure the first and the second register both, then there is a high probability that we will get $|\tilde{\phi}_j\rangle |u_j\rangle$ as the output for some j . In this way, we would have determined the phase of eigenvalue corresponding to some eigenvector. Repeating this many times, we can get phase for all the eigenbases!!

7.2 Order-finding

7.2.1 Problem

This, along with the Shor's Algorithm, are one of the most useful and celebrated algorithm in the quantum computing realm till now. The problem is quite simple: We have to find the order r of a given positive integer x with respect to a given N . The order of a number $x(< N)$ is defined as the smallest number r such that

$$x^r = 1(\text{mod } N) \quad (7.5)$$

The above statement requires that $\gcd(x, N) = 1$ which can be easily checked using the Euclid's Long Division Algorithm.

7.2.2 Classical Solution

The classical solution for this problem is to compute x^r for all r sequentially from 1 to N . This requires $O(N)$ steps and hence the time complexity is $O(N)$.

7.2.3 Quantum Solution

In this problem also like the previous ones, we get an exponential speedup. We use the quantum phase estimation we defined in the previous problem to achieve this speedup.

So, for the quantum case, we have two inputs: x and N both of which can be expressed using $L = \log(N)$ qubits. This is used to make an oracle which performs the following operation:

$$U|y\rangle = |xy(\text{mod } N)\rangle \quad (7.6)$$

where $y \in \{0, 1\}^L$. We have to find the eigenvectors and eigenvalues of this operator.

We know that $U|x^k\rangle = |x^{k+1} \text{ mod } N\rangle$ and $U|x^{r-1}\rangle = |1 \text{ mod } N\rangle$ where r is the order. So, we will combine these various states to get our eigenvectors. So, let's say the eigenvectors are given by

$$|u_s\rangle = \sum_{k=0}^{r-1} a_{k,s} |x^k \text{ mod } N\rangle \quad (7.7)$$

This gives us the constraint that $a_{k-1,s} = \lambda_s a_{k,s}$ (for $k = 0$, the LHS would be $a_{r-1,s}$) where λ_s are the eigenvalues and hence $\lambda_s = e^{f_r(s)}$. By simple calculations we can show that

$$\lambda_s = e^{2\pi i s/r} a_{k,s} = \frac{e^{-2\pi i s k/r}}{\sqrt{r}} \quad (7.8)$$

So the eigenvectors are

$$|u_s\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} e^{-2\pi i s k/r} |x^k \text{ mod } N\rangle \quad (7.9)$$

Notice that the phase of eigenvalues is s/r . So, we can use phase estimation to get the value of s/r from which we can extract r with a bit of work.

Note that to use phase estimation we must be able to compute U^{2^j} . For this we use modular exponentiation. Let us say that the content of first register is $|z\rangle$ and second one is $|y\rangle$. Then the first part of phase estimation is:

$$\begin{aligned} |z\rangle|y\rangle &\rightarrow |z\rangle U^{z_t 2^{t-1}} \dots U^{z_1 2^0} |y\rangle \\ &= |z\rangle |x^{z_t 2^{t-1}} \times \dots \times x^{z_1 2^0} y(\text{mod } N)\rangle \\ &= |z\rangle |x^z y(\text{mod } N)\rangle \end{aligned} \quad (7.10)$$

So, we first compute the various powers of x i.e. $x^{2^j}(\text{mod } N)$ and store them in a separate register. This can be accomplished in $O(L^3)$ steps by repeatedly squaring in $O(L^2)$ for $O(L)$ values of j . Then we perform the multiplication to get $x^zy(\text{mod } N)$ again in $O(L^3)$ steps. So, in total, this step needs $O(L^3)$ steps.

The other issue is that we need to have an eigenvector which depends on the order. To tackle this we give a superposition of the eigenvectors as input in the second register and as mentioned earlier this does not effect the result much and we will get a good estimate of s/r for some s with great probability.

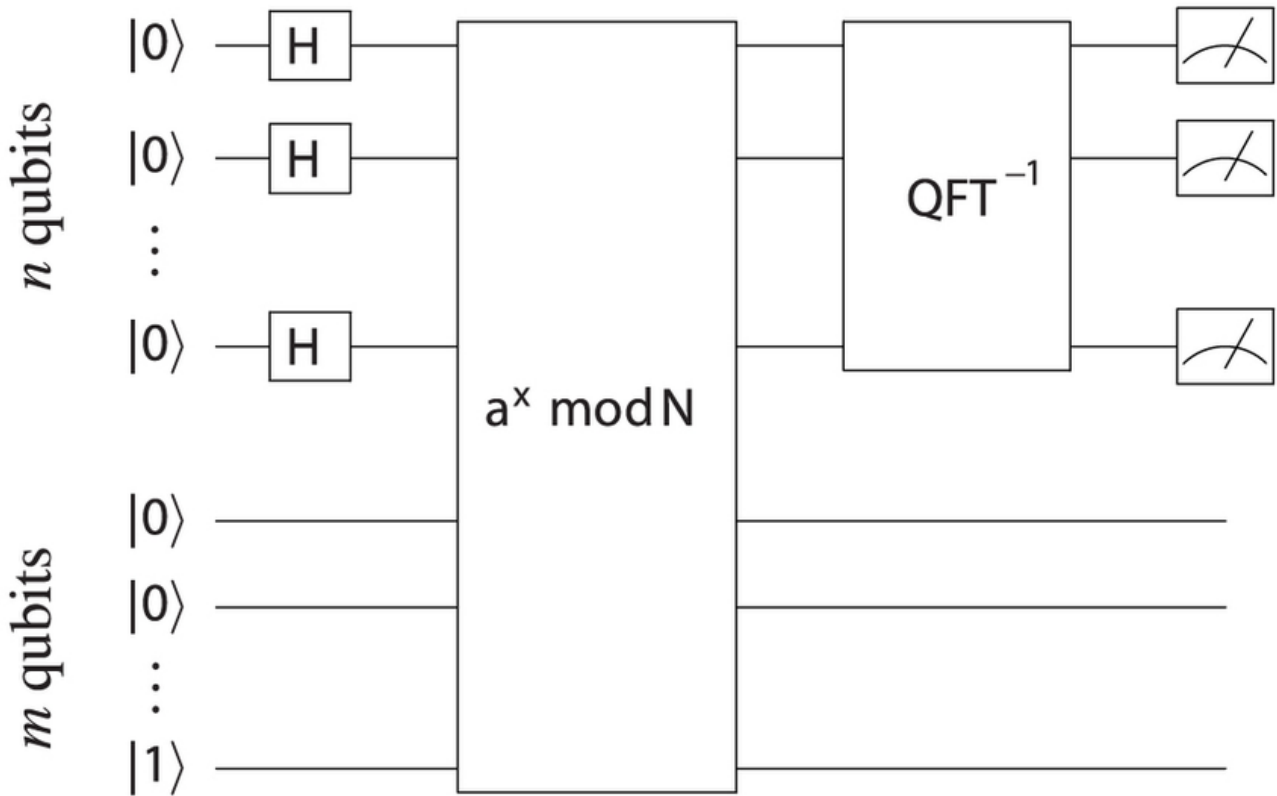


Figure 7.2: Circuit for finding the order. In this case we find order of a w.r.t. N

To get r from s/r we use the method of continued fraction to obtain a good estimate of it with the error less than $1/2r^2$. Once we have approximated it, we can get a number r' which will be a candidate for the order. We can easily verify if it is correct, and if it is we are done. If it is not, we will have to repeat the process again.

7.3 Period Finding(Shor's Algorithm)

7.3.1 Problem

As the name suggests, given a function $f : \{0, 1\}^L \rightarrow \{0, 1\}$ which is periodic with period r , we have to find the value of the period r .

The algorithm to solve this is known as the Shor's Algorithm. The famous application of quantum computing to break the RSA cryptosystem is based on this algorithm.

7.3.2 Classical Solution

The classical solution includes searching for the period linearly through all the values of r possible. There will be 2^L values to be checked where L is the size of input. Also, each pass would require $O(2^L)$ steps to verify whether it is the period or not. So, in total it is a $O(2^L)$ algorithm which is very-inefficient for even L in the range of thousands.

7.3.3 Quantum Solution

As we have seen till now, the use of quantum computer allows an exponential speed up. For the quantum case, we need an oracle which would perform the following operation:

$$U|x\rangle|y\rangle \rightarrow |x\rangle|y \oplus f(x)\rangle \quad (7.11)$$

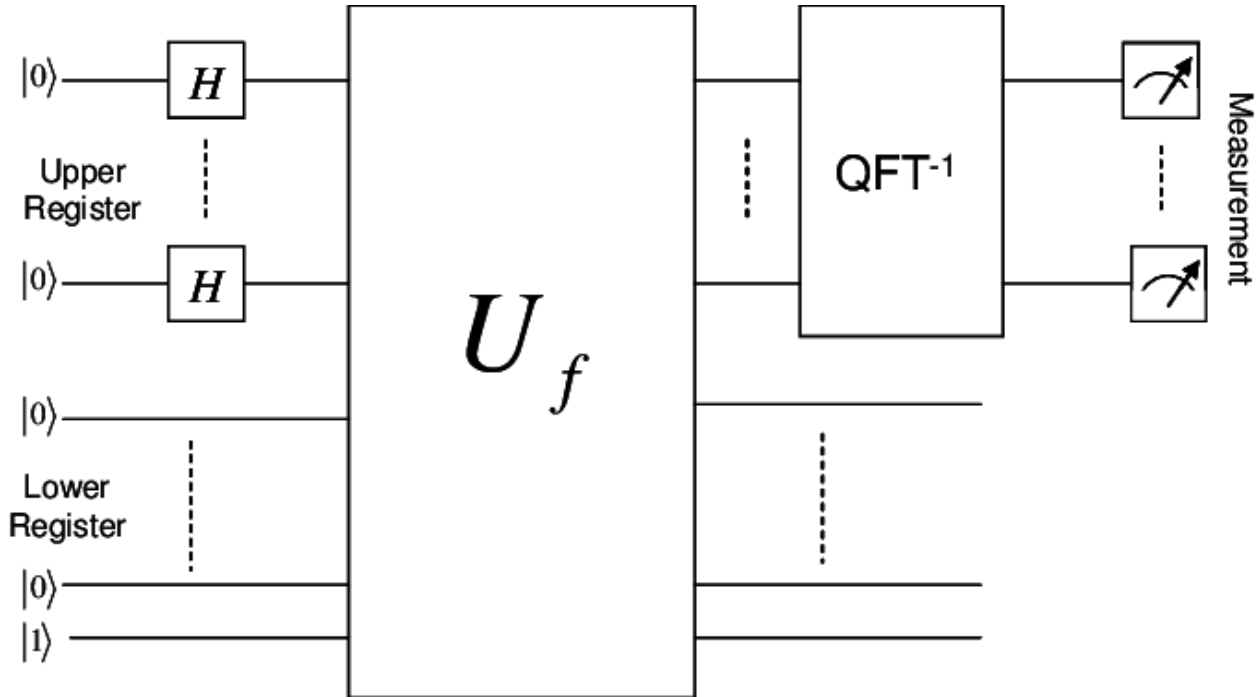


Figure 7.3: Circuit for finding the period of a given function

The algorithm is similar to the previous ones. We first transform the input to get the period in the phase and then use inverse QFT. The input consists of two registers similar to the phase estimation one. The first one is of size t and is used to store the period while the second one is used to store $f(x)$ and is of size 1.

1. The input is taken to be $|0\rangle|0\rangle$.
2. Hadamard gate is used on the first register to get a superposition of all the states. So, the state becomes

$$\frac{1}{\sqrt{2^t}} \sum_{x=0}^{2^t-1} |x\rangle|0\rangle$$

3. Then we apply the oracle to get the value of $f(x)$ in the second register.

$$\frac{1}{\sqrt{2^t}} \sum_{x=0}^{2^t-1} |x\rangle|f(x)\rangle$$

Note that the values of $f(x)$ will have a Fourier Transform say $\hat{f}(l)$ given by

$$|\hat{f}(l)\rangle \equiv \frac{1}{\sqrt{r}} \sum_{x=0}^{r-1} e^{-2\pi i l x / r} |f(x)\rangle \quad (7.12)$$

and so

$$|f(x)\rangle = \frac{1}{\sqrt{r}} \sum_{l=0}^{r-1} e^{2\pi i l x / r} |\hat{f}(l)\rangle \quad (7.13)$$

Substituting this above we get the state as

$$\frac{1}{\sqrt{r}2^t} \sum_{l=0}^{r-1} \sum_{x=0}^{2^t-1} e^{2\pi i l x / r} |x\rangle |\hat{f}(l)\rangle$$

4. If we combine the phase factor with the first register, we see that it is the inverse Fourier transform of l/r . So, we apply QFT^\dagger to get $|\widetilde{l/r}\rangle$.
5. Measure the first register to get the value of $\widetilde{l/r}$.
6. Apply continued fraction algorithm to get the value of r .

So, we get the value of r . The algorithm requires $O(L)$ gates to calculate the period. Next we use it to calculate the factors of a number.

7.4 Factorization

7.4.1 Problem

Given a number N , we have to find some factor of N .

7.4.2 Classical Solution

The classical solution requires to check if any number from 1 to \sqrt{N} divides N . So, in total we will be checking \sqrt{N} numbers and hence the time complexity is $O(\sqrt{N})$.

7.4.3 Quantum Solution

For the quantum solution, we assume a specific class of N . First, it must not be a multiple of 2 which can be easily checked. Second, the number must not be of the form $a^b \forall b \in \{2, 3, 4 \dots \log N\}$. This can be easily checked in $O((\log N)^3)$ steps by computing a for all values of b .

The algorithm first reduces the problem of factorization to order finding and then use the algorithm for order finding to find the factor. It uses the following theorem:

THEOREM 1: Suppose N is an L bit composite number, and $x \leq N$ is a non-trivial solution of $x^2 = 1 \pmod{N}$ then either $\gcd(x - 1, N)$ or $\gcd(x + 1, N)$ is a non-trivial factor of N .

THEOREM 2: Suppose $N = p_1^{\alpha_1} p_2^{\alpha_2} \dots p_m^{\alpha_m}$ is the prime factorization such that all p_i are odd i.e. N is not even. Let x be a randomly chosen number less than N and co-prime to N and let r be the order of x w.r.t N . Then,

$$P(r \text{ is even and } x^{r/2} \not\equiv -1 \pmod{N}) \geq 1 - \frac{1}{2^m} \quad (7.14)$$

This probability is always greater than half and hence repeated application will take it close to 1. Combining all these facts, we can efficiently compute some factor of a given composite number.

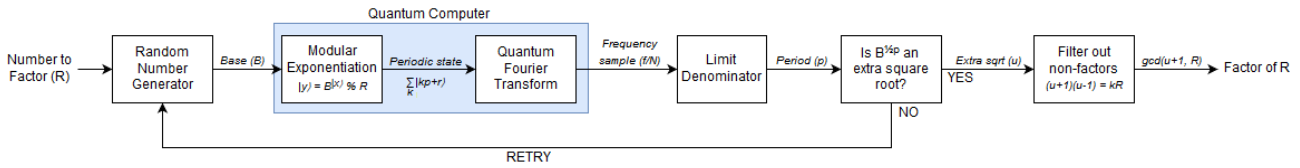


Figure 7.4: Circuit for finding a factor of a given number N

1. Ensure that N is odd. If not then return 2 as a factor. This can be done in $O(1)$ steps.
2. Ensure that N is not of the form a^b otherwise return a . This can be done in $O(L^3)$ steps.
3. Randomly choose a number $x \in \{0, 1, 2 \dots N - 1\}$. If x is not co-prime to N , then return $\gcd(x, N)$ as a factor.
4. Since x is co-prime to N , there would exist an order of x w.r.t. N say r . Compute thus r using the order finding circuit.

5. If r is even and $x^{r/2} \not\equiv -1 \pmod{N}$ then compute $\gcd(x^{r/2} - 1, N)$ and $\gcd(x^{r/2} + 1, N)$. If either one of them is a non-trivial factor, return it. Otherwise, the algorithm failed and repeat for a different value of x .

In all, we require $O(L^3)$ operations to calculate a factor of the number N .

hence we have found a way to calculate the factors of a given number N . The inability of computing such factorization for large numbers is one of the key points in the modern cryptosystems like RSA. If a quantum computer is built which is able to perform the above algorithm, these systems will be broken. hence quantum computer poses a great threat to our privacy until we find some other stronger system.

Chapter 8

Quantum Search

The quantum search algorithm is another set of algorithms that can offer a speed up by using a quantum computer. It offers a speed-up of square root. Though it is not as great as the exponential speed up in case of quantum Fourier transform but still it is used for a large variety of problems and can offer speed up for the NP problems.

The search problem may include a very large variety of problem. The normal search problems like searching for a number in a list of numbers can be said to have a structure. But the quantum search algorithm is much more general and can be used for larger variety of problems which may not even have a structure.

8.0.1 Oracle

For the search problem, we assume a very general structure. Let us say we have N elements indexed from 0 to $N - 1$. Also we have a function $f(x)$ which verifies whether x is a solution of the search problem or not i.e. it returns 1 if x is a solution otherwise it returns 0. Note that $f(x)$ just verifies whether a given index is a solution of the search problem or not. It does not search for the solution.

We assume that we have a circuit which can implement the function $f(x)$. So, let us say that it is O . Then

$$O|x\rangle|y\rangle = |x\rangle|y \oplus f(x)\rangle \quad (8.1)$$

8.0.2 Solution(Grover's Algorithm)

For the solution of quantum search problem, we need to apply the oracle described above repeatedly. Along with that we need one more operator which changes the phase of all the bases vector except the $|0\rangle$. So, it takes $|x\rangle$ to $-|x\rangle$ for all $x > 0$. This can be written as

$$P = 2|0\rangle\langle 0| - I$$

Also, we need the number of solution M but this can be calculated using phase estimation.

We take two registers- one is the output register and the second one is the oracle workspace register initialized to $(|0\rangle - |1\rangle)/\sqrt{2}$. The value of second register remains unchanged throughout the process. When we apply the oracle, the state is changed to $(-1)^{f(x)}|x\rangle(|0\rangle - |1\rangle)/\sqrt{2}$ and so only the phase of first register changes dependent on the value of $f(x)$.

We define the Grover operator as:

1. Apply the oracle O .
2. Apply H on the first register.

3. Apply P gate on the first register.

4. Apply H gate again on the first register.

The last three steps combine to give an operator $2|\psi\rangle\langle\psi| - I$ where $|\psi\rangle = \sum |x\rangle$. This operator gives a flips the space about the $|\psi\rangle$ vector.

The first step flips the space about the space of vectors which are not the solutions of the search problem. In all, all the four steps shift the initial state towards the solution of the search problem.

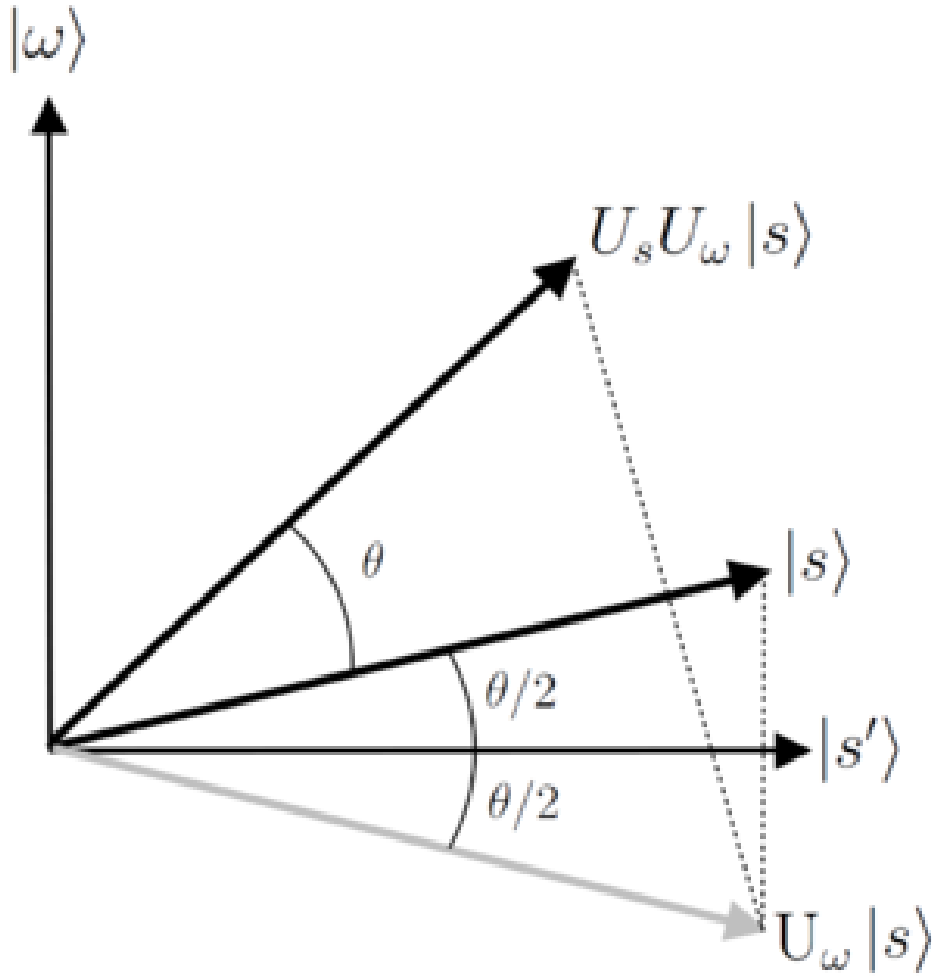


Figure 8.1: Phase diagram for Grover Iteration

In the above diagram, $|\omega\rangle$ is the space of the solution while $|s'\rangle$ is the space of other vectors. So, the state turns towards the solution by an angle θ given by:

$$\cos(\theta/2) = \sqrt{\frac{(N-M)}{N}} \quad (8.2)$$

and after k iterations of the Grover Algorithm

$$G^k|\psi\rangle = \cos\left(\frac{2k+1}{2}\theta\right)|s'\rangle + \sin\left(\frac{2k+1}{2}\theta\right)|\omega\rangle \quad (8.3)$$

So, the number of iterations before reaching the solution space is $k = \left\lfloor \frac{\arccos\sqrt{M/N}}{\theta} \right\rfloor$.

Note that if the value of M is greater than $N/2$, then the value of k rises with M . So, we

append extra N numbers at the end which are not the solution to make it less than $N/2$. K is in the range of $O(\sqrt{N})$. So, we get a quadratic speedup in this case. The circuit used is shown below:

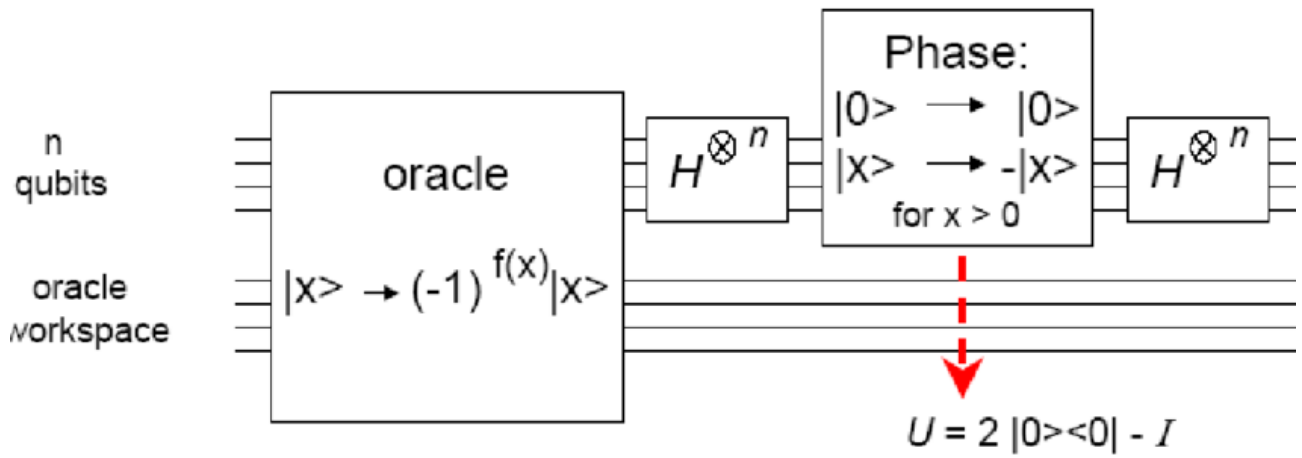


Figure 8.2: Circuit for Grover Iteration

The above circuit is repeated for k number of times, to get the correct answer.