Quantum Teleportation Algorithm

Rutvij Menavlikar
2019111032

November 23, 2020

Abstract

This document is an introduction to the Quantum teleportation protocol and some of the concepts associated with it.

Contents

1 The need for a separate Teleportation Protocol

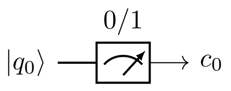
Given two Quantum Computers, where one of these computers has a qubit in some qubit state, and now it wants to transfer this qubit state to the other computer.

For a classical computer, this procedure is pretty straightforward. You measure the bit, which can be in exactly two states, i.e. **0** and **1**, and send this data over to the other computer via some network. And after receiving this data, the other computer can construct it's bit identical to the bit of the first computer.

But the problem in quantum computers is that a qubit can exist in infinite states($\mathbb{C}^2$), but measuring that qubit destroys it's state and reduces it to exactly one of 2 orthogonal qubit states.
Let the qubit to be teleported be q_0, and let

$$|q_0\rangle = \alpha |0\rangle + \beta |1\rangle \text{ where } \alpha, \beta \in \mathbb{C}$$

Now measuring the qubit q_0, and storing the result in the classicla bit c_0,



The bit c_0 exists in only two state, i.e. **0**(with probability $|\alpha|^2$) and **1**(with probability $|\beta|^2$).
The initial qubit q_0 has been destroyed and since infinite number of states have been stored as one of two states, data is lost and it is impossible to rebuild the same qubit in the other quantum computer.
Also, note that creating the qubit repeatedly and measuring it to get $|\alpha|^2$ and $|\beta|^2$ is not enough either. Because for a complex number z, even when we know $|z|$, there are infinite possibilities for the value of z.

Hence, measuring and sending over the results of measurement will not work every time.

2 No Cloning Theorem

Another need for a Quantum Teleportation Protocol is the No Cloning Theorem.
In classical computers we can easily copy the state of 1 bit to another bit without measuring the bit and creating the state in another bit. But, this is not the case in a quantum computer. This operation is prevented by the No Cloning theorem.

Theorem 1 (No Cloning Theorem)**.** *It is impossible to create an independent and identical copy of an arbitrary unknown quantum state.*

Proof. First, we define inner product of two quantum states $|\psi\rangle$ and $|\phi\rangle$ as $\langle\psi|\phi\rangle$, where

$$|\psi\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle\,,\ |\phi\rangle = \beta_0 |0\rangle + \beta_1 |1\rangle \text{ and } \alpha_0, \alpha_1, \beta_0, \beta_1 \in \mathbb{C}$$

then,

$$\langle\psi|\phi\rangle = \overline{\alpha_0}\beta_0 + \overline{\alpha_1}\beta_1$$

where for a complex number z, $\overline{z}$ is it's complex conjugate.
We also observe that,

$$\begin{aligned}\langle\psi\psi|\phi\phi\rangle &= \langle|\psi\rangle \otimes |\psi\rangle||\phi\rangle \otimes |\phi\rangle\rangle \\ &= \big\langle(\alpha_0^2 |00\rangle + \alpha_0\alpha_1 |01\rangle + \alpha_0\alpha_1 |10\rangle + \alpha_1^2 |11\rangle)\big|(\beta_0^2 |00\rangle + \beta_0\beta_1 |01\rangle + \beta_0\beta_1 |10\rangle + \beta_1^2 |11\rangle)\big\rangle \\ &= \overline{\alpha_0^2}\beta_0^2 + \overline{\alpha_0\alpha_1}\beta_0\beta_1 + \overline{\alpha_0\alpha_1}\beta_0\beta_1 + +\overline{\alpha_1^2}\beta_1^2 \\ &= (\overline{\alpha_0}\beta_0 + \overline{\alpha_1}\beta_1)^2 \\ &= \langle\psi|\phi\rangle^2\end{aligned}$$

Let us assume that $|\psi\rangle$ and $|\phi\rangle$ are not orthogonal, i.e. $\langle\phi|\psi\rangle \neq 0$
If possible, there exists a unitary transformation U, such that for any quantum state $|\phi\rangle$, $U|\phi\rangle \otimes |0\rangle = |\phi\rangle \otimes |\phi\rangle$
Now, we have

$$U|\psi\rangle \otimes |0\rangle = |\psi\rangle \otimes |\psi\rangle \text{ and } U|\phi\rangle \otimes |0\rangle = |\phi\rangle \otimes |\phi\rangle$$

Now we consider

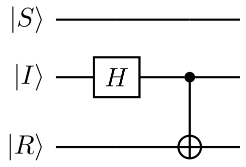
$$\begin{aligned}\langle\psi|\phi\rangle &= \langle|\psi\rangle \otimes |0\rangle||\phi\rangle \otimes |0\rangle\rangle \\ &= (|\psi\rangle \otimes |0\rangle)(|\phi\rangle \otimes |0\rangle) \\ &= (|\psi\rangle \otimes |0\rangle)U\text{`}\ .\ U(|\phi\rangle \otimes |0\rangle) \qquad &&\text{(U is a unitary transformation,} \\ & &&\text{and hence it is reversible)} \\ &= (|\psi\rangle \otimes |\psi\rangle)(|\phi\rangle \otimes |\phi\rangle) \\ &= \langle\psi|\phi\rangle^2\end{aligned}$$

And since $|\psi\rangle$ and $|\phi\rangle$ are not orthogonal, $\langle\psi|\phi\rangle = 1$.
And this is not the case for every pair of quantum states $|\psi\rangle$ and $|\phi\rangle$.
Contradiction

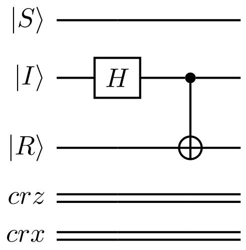
Hence, it is impossible to create an independent and identical copy of an arbitrary unknown quantum state. □

3 Pre-requisites to Quantum Teleportation

Suppose the sender quantum computer has a qubit $|S\rangle$ in an unknown quantum state, and we want to transfer this quantum state to the receiver quantum computer's qubit $|R\rangle$.
We would need to set $|R\rangle$ to $|0\rangle$ and the sender computer would require a qubit $|I\rangle$, when $|I\rangle$ and $|R\rangle$ are in a bell state, i.e. $|IR\rangle = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle$.

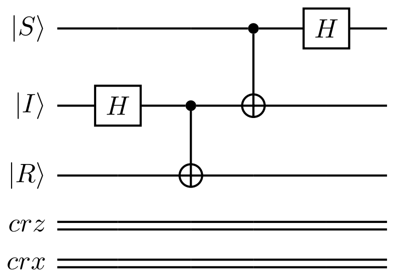


We will also require two classical bits crx and crz which will be sent from the sender quantum computer to the receiver quantum computer.
Hence, the circuit before applying the teleportation protocol will be



4 The Quantum Teleportation Protocol

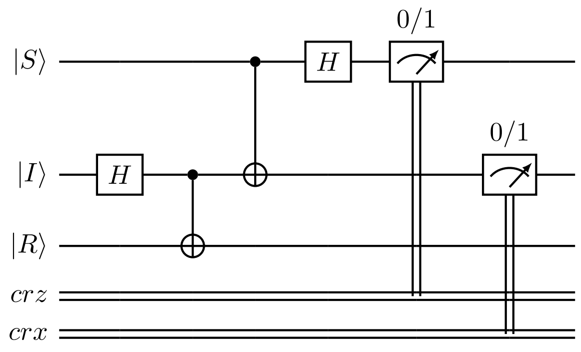
Now that we have the two classical bits and the three qubits, two of which are in a bell state, we are ready to implement the teleportation protocol.
Note that because of the No Cloning theorem, when we teleport a qubit, we destroy it's quantum state in one place and recreate the exact same quantum state on another qubit.
We first apply a CNOT gate with control and target qubits as $|S\rangle$ and $|I\rangle$ respectively, and then we apply a Hadamard gate to the qubit $|S\rangle$.



If we assume that $|S\rangle = \alpha|0\rangle + \beta|1\rangle$ where $\alpha, \beta \in \mathbb{C}$, then after these operations we get the combined state of the qubits as

$$|SIR\rangle = \frac{\alpha}{2}|000\rangle + \frac{\beta}{2}|001\rangle + \frac{\beta}{2}|010\rangle + \frac{\alpha}{2}|011\rangle + \frac{\alpha}{2}|100\rangle - \frac{\beta}{2}|101\rangle - \frac{\beta}{2}|110\rangle + \frac{\alpha}{2}|111\rangle$$

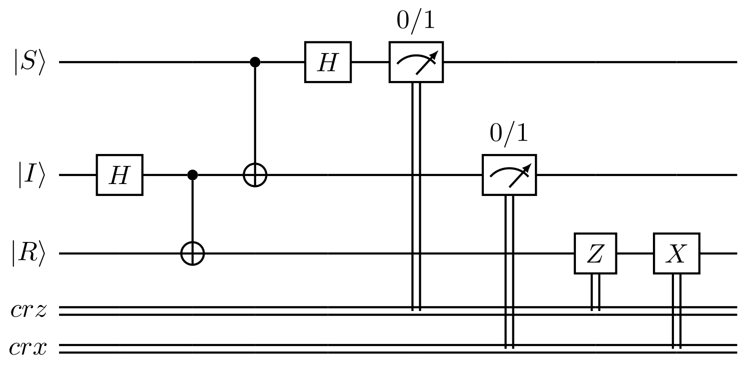
We now measure the qubits $|S\rangle$ and $|I\rangle$ and store the result in the classical bits crz and crx respectfully.



The following table denotes the state of $|R\rangle$ for each value of (crz, crx).

crz	crx	$\|R\rangle$
0	0	$\alpha\|0\rangle + \beta\|1\rangle$
0	1	$\beta\|0\rangle + \alpha\|1\rangle$
1	0	$\alpha\|0\rangle - \beta\|1\rangle$
1	1	$\alpha\|1\rangle - \beta\|0\rangle$

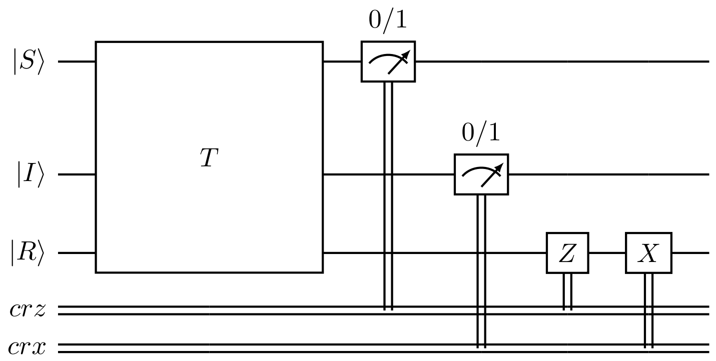
To rebuild $|R\rangle$ to the original value of $|S\rangle$, we can use the calculated values of crz and crx as conditions to apply the Z gate and X gate on $|R\rangle$.



5 Implementation of Quantum Protocol

The code for this protocol has been written in qiskit and a script to simulate the teleportation using the QASM simulator have been included in the project files.

The entanglement in bell state and then further operations of CNOT and Hadamard gate can all be replaced by one 3 bit quantum gate T on which measurements are done to complete the teleportation protocol.



In the case where *qubit*0 is $|S\rangle$, *qubit*1 is $|R\rangle$ and *qubit*2 is $|I\rangle$.
When *qubit*0 is in the state (Bloch Sphere notation)

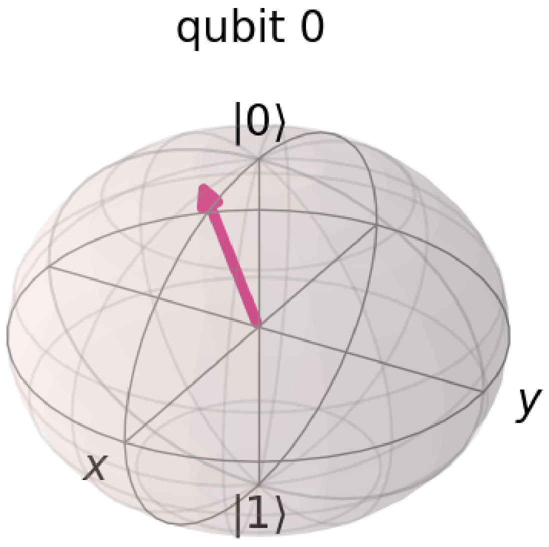


Figure 1: Initial Quantum state of sender quantum computer's qubit

After completing the teleportation the protocol, the final states of the qubits are (in Bloch Sphere notation)

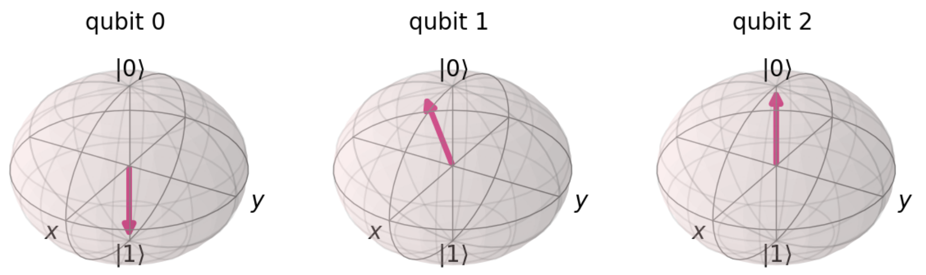


Figure 2: Final Quantum state of the sender quantum computer's qubit, the intermediate qubit and the receiver quantum computer's qubit

And as you can observe, VOILA! We have achieved Quantum Teleportation!

6 An Important Note

Given the fact that when we measure the qubits $|S\rangle$ and $|I\rangle$, the effects of it are immdiately observed on $|R\rangle$, one may assume that this protocol allows communications to take place faster than the speed of light.
But this is not accurate.

While the quantum state of $|R\rangle$ may be updated instantaneously, and may also have happened before light would have arrived to it's location, it is impossible to exactly determine which of the four possible quantum states to use without knowing the two classical bits.
And these Classical bits arrive at a speed less than or equal to the speed of light.
Hence, Quantum Teleportation Algorithm cannot achieve communication faster than the speed of light.

BB84 Protocol

Sriram Devata
2019113007

Algorithm Analysis And Design

Abstract

This document gives an overview of the BB84 protocol for the implementation of Quantum Key Distribution(QKD) and the relevant principles needed to understand it.

Contents

1 Introduction

All cryptography algorithms depend on secure sharing of keys. Though symmetrical cryptography algorithms are faster over asymmetrical cryptography algorithms, the risk of the shared key being intercepted makes them an inferior choice to an asymmetrical algorithm.

When two parties (for simplicity, assume these parties to be Alice and Bob) are trying to establish communication by sharing keys to encrypt future information exchange, there's a risk of an eavesdropper (for simplicity, assume the eavesdropper is called Eve) intercepting the communication and figuring out the key that Alice and Bob decide to use. Besides having no point of encoding and decoding the information, Alice and Bob are not aware if their key has been compromised and would have the false belief that they have established a secure line of communication.

Using the BB84 protocol, Alice and Bob will know if Eve is eavesdropping and will not use the compromised key for further communication.

2 Background

2.1 Qubits

In any computational devices, a building block is a two-state system (0, 1). In quantum mechanics, any system can exist in a *superposition* of possible states. A qubit is a 2-state quantum system. It can exist in either of the possible states, or in a superposition of the two states. In general, the state of a qubit can be described as:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle, \text{ with } \alpha, \beta \in C \text{ and } |\alpha|^2 + |\beta|^2 = 1$$

2.2 Measurement of a Qubit

The superposition $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ is the *private* information of a qubit. In order for us to know the state of the qubit, we have to make a measurement. When we make a measurement, we end up with one bit of information - if the qubit collapsed to the state $|0\rangle$ or $|1\rangle$. It collapses to the state $|0\rangle$ with probability $|\alpha|^2$ and collapses to the state $|1\rangle$ with probability $|\beta|^2$.

Once the qubit collapses to a state after we make a measurement, the superposition of the qubit is lost and it now exists in the collapsed state. This means that it is not possible to find the values $|\alpha|^2$ and $|\beta|^2$ by making multiple measurements on a qubit to get a probability distribution.

A qubit can be visualized as a vector on a plane. The state of any qubit can be expressed as a linear combination of two orthogonal states. Two standard basis sets are the Z-basis states $\{|0\rangle, |1\rangle\}$ and the X-basis states$\{|-\rangle, |+\rangle\}$.

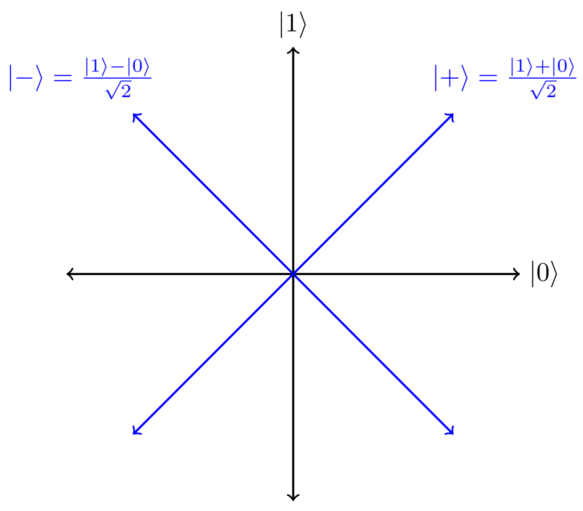

$|1\rangle$
$|-\rangle = \frac{|1\rangle - |0\rangle}{\sqrt{2}}$
$|+\rangle = \frac{|1\rangle + |0\rangle}{\sqrt{2}}$
$|0\rangle$

Figure 1: The Standard(Z) basis and the Signed(X) basis

In general, we can choose any two orthogonal states and express the state of a qubit as a linear combination of these two states.

2.3 Measurements In Different Bases

A central component of the BB84 protocol is the collapse of the qubit when it is measured in a basis different from the basis it was encoded in. For example, if the bit 0 was encoded onto a qubit in the Z basis and is measured in the X basis, the result of the measurement would be 0 or 1 with equal probability.

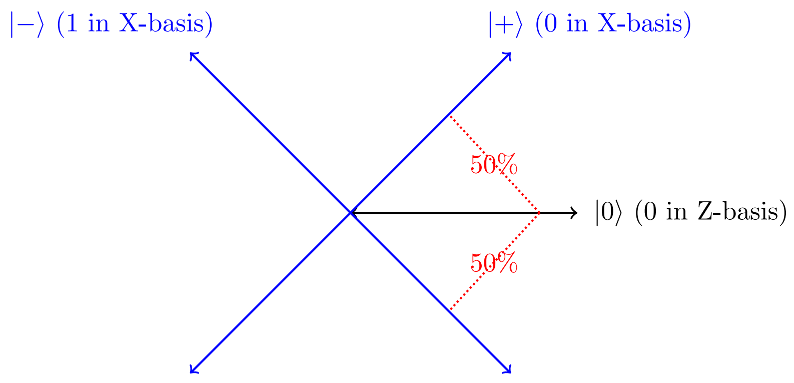

$|-\rangle$ (1 in X-basis)
$|+\rangle$ (0 in X-basis)
50%
$|0\rangle$ (0 in Z-basis)
50%

Figure 2: Measuring Z-basis 0 in the X-basis

2.4 Quantum Communication Channel

Classical communication channels can transmit classical information. Bits can be encoded as high or low voltages. This cannot be used sending and receiving qubits. To transmit quantum information, BB84 utilizes a separate Quantum communication channel to send and receive the states of qubits. The practical implementation of such a communication channel can be a fibre-optic cable that conserves the polarisation of photons. The state of a qubit can be encoded in the polarization of a photon. We can map the possible polarization states of a photon to the vector representation of a qubit.

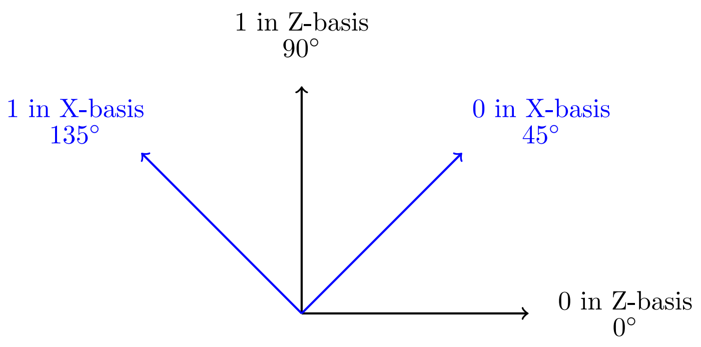


Figure 3: Mapping the Polarization of a Photon to Qubit states

In this particular quantum communication channel, the bits encoded in different bases and the resulting qubits can be shown as $\nwarrow, \uparrow, \nearrow$ and $\rightarrow$.

Bit to encode	0						1					
Encoding Basis	Z			X			Z			X		
Photon Polarization	$\rightarrow$			$\nearrow$			$\uparrow$			$\nwarrow$		
Measurement Basis	Z	X		Z		X	Z	X		Z		X
Measured Bit	0	0	1	0	1	0	1	0	1	0	1	1

Table 1: Qubits when encoded and measured in different bases

3 Overview of the Procotol

The protocol initially assumes that there are two channels(quantum and classical) of communication available to Alice and Bob, and Eve is able to intercept information in both the channels. Alice chooses a random string of bits, and decides to encode each of these bits as a qubit, in either the Z-basis or X-basis at random. She keeps track of the initial bit string and the set of bases she chose to encode each bit. She sends the resulting qubits over to Bob through the quantum channel.

As Bob receives the qubits, he measures each of the qubit in either the Z-basis or X-basis at random. He keeps track of the bits he measures and the bases he chose to measure each qubit. Bob now sends the set of bases he chose to measure each qubit and sends it to Alice. Alice responds to this by telling Bob the qubits for which Bob chose the same bases to measure in. Alice and Bob discard the bits where they both chose different bases to encode and measure.

Alice and Bob take a small subset of the bits and compare them. Assuming that there is proper noise correction and there is no measurement errors, if Eve didn't eavesdrop, Alice and Bob would end up with the same bits. They discard this subset of bits, and use the rest of the bits to form a secret key for further communication.

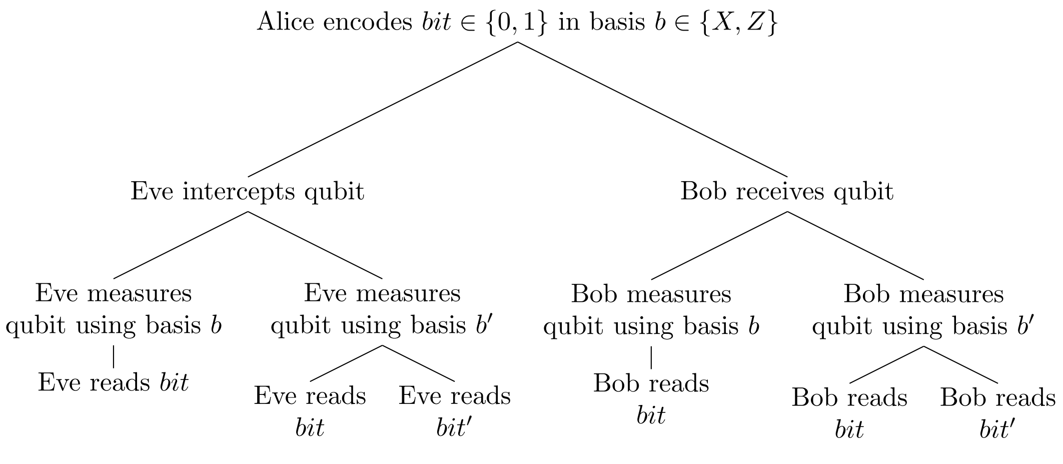
If the small subset of bits do not agree with each other, it would indicate that an eavesdropper, Eve, has intercepted the message, and the communication isn't secure. This is because if Eve wants to figure out the key, she would have to make measurements on the qubits sent by Alice in either of the bases chosen at random. According to the no cloning theorem, she can't make a replica of Alice's qubit to forward to Bob and make a measurement on her copy of the qubit. She either has to send the qubit she has measured, or guess the basis and send a new encoded qubit to Bob.

4 Simulation Code

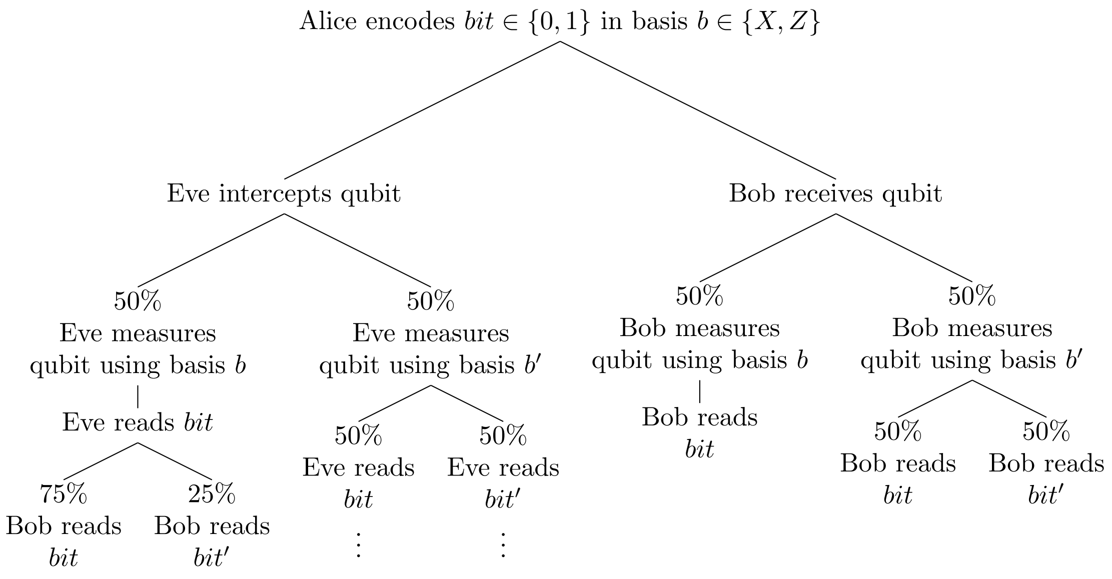
An implementation of the BB84 protocol in Qiskit can be found as an interactive example here.

5 Analysis

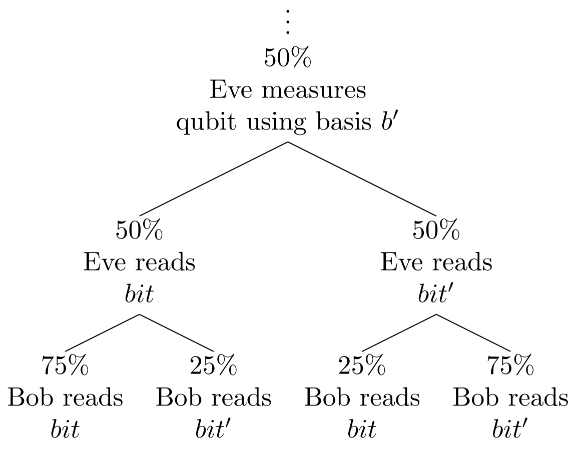
For each bit that Alice encodes, the following flowchart illustrates the bit that Bob will measure if Eve doesn't intercept the qubit, and the bit that Eve measures if Eve intercepts the qubit.


Alice encodes $bit \in \{0,1\}$ in basis $b \in \{X, Z\}$
Eve intercepts qubit
Bob receives qubit
Eve measures qubit using basis b
Eve reads bit
Eve measures qubit using basis b'
Eve reads bit
Eve reads bit'
Bob measures qubit using basis b
Bob reads bit
Bob measures qubit using basis b'
Bob reads bit
Bob reads bit'

Since both Eve and Bob choose the basis to measure in randomly, there is a 50% chance that they choose the right basis. If they choose the wrong basis, there is a 50% chance that they measure the correct bit. In the BB84 protocol implementation attached with this document, Eve forwards the qubit that she already made the measurement on to Bob. This means that in the case when Eve chose the wrong basis and the qubit collapsed to bit', Bob measures bit and bit' with different probabilities.


Alice encodes $bit \in \{0,1\}$ in basis $b \in \{X, Z\}$
Eve intercepts qubit
Bob receives qubit
50% Eve measures qubit using basis b
Eve reads bit
75% Bob reads bit
25% Bob reads bit'
50% Eve measures qubit using basis b'
50% Eve reads bit
50% Eve reads bit'
50% Bob measures qubit using basis b
Bob reads bit
50% Bob measures qubit using basis b'
50% Bob reads bit
50% Bob reads bit'

In the case where Eve intercepts the qubit and measures using the basis b',



Such an analysis will help us to choose optimal initial bit string length and the subset of the bit string to choose, to bring the number false negatives of an eavesdropper within the error threshold.

Simons's Algorithm Tutorial

Ishaan Shah

November 2020

1 The Problem

Given a function (implemented as a black box) $f : \{0,1\}^n \rightarrow \{0,1\}^n$, promised to satisfy the property that, for some non zero binary string $s \in \{0,1\}^n$

$$f(x) = f(y) \Leftrightarrow x = y \text{ or } x = y \oplus s$$

Find the secret string s in least possible queries. It is important to note that the function $f(x)$ given to us is in the form of a black box which means we cannot extract the bit string by looking at the circuit inside.

For example consider the following truth table for $n = 2$ and $s = 10$

x	$f(x)$
00	10
01	01
10	10
11	01

2 Algorithm

2.1 The classical way

The classical algorithm is pretty straightforward. We find the output for 0^n and store it. Then we find outputs of successive binary strings until we find the output we got for 0^n. The input for which this is true will be the secret bit string. We can easily see that the time complexity for this is $\mathcal{O}(2^n)$ and space complexity is $\mathcal{O}(1)$.

2.2 The quantum way

Before we begin to understand how Simon's algorithm works, we have to look at a smaller sub-algorithm namely the **Fourier Sampling**.

Fourier Sampling

Fourier sampling is the process of applying n Hadamard Gates to a system of n qubits and measuring the result.

If the input to the circuit is denoted by $|\psi\rangle$ and then the output $|\psi'\rangle$ is given by

$$|\psi'\rangle = \sum_{y\in\{0,1\}^n} \frac{(-1)^{y\cdot\psi}}{2^{n/2}} |y\rangle$$

Now that we know how Fourier sampling works lets move on to the actual algorithm. The actual algorithm consists of three steps

2.2.1 Step 1 - Superposition

In the first step, we create an equal superposition of all the possible bit strings of length n. This can easily be done by initialising n qubits to 0 and passing them through n Hadamard gates. So now the state of our n qubits will be given by

$$|\psi\rangle = \sum_{y\in\{0,1\}^n} \frac{1}{2^{n/2}} |y\rangle$$

2.2.2 Step 2 - Apply $f(x)$ and measure

After creating the superposition described below, we pass the vector $|\psi\rangle$ into the black box provided to us. Now we measure the output qubits that we get from the black box which causes the input vector $|\psi\rangle$ to collapse to the following state

$$|\psi\rangle = \frac{1}{\sqrt{2}} |z\rangle + \frac{1}{\sqrt{2}} |z \oplus s\rangle \qquad \text{...where } z \in \{0,1\}^n$$

2.2.3 Step 3 - Fourier sampling

Now we apply the Fourier sampling algorithm to the input which is in the above mentioned state. After applying n Hadamard gates to the n input bits the state of the vector $|\psi\rangle$ becomes

$$\begin{aligned}|\psi\rangle &= \sum_{y\in\{0,1\}^n} \left(\frac{(-1)^{y\cdot z} + (-1)^{y\cdot(z\oplus s)}}{2^{(n+1)/2}}\right) |y\rangle \\ &= \sum_{y\in\{0,1\}^n} \left(\frac{(-1)^{y\cdot z}[1 + (-1)^{y\cdot s}]}{2^{(n+1)/2}}\right) |y\rangle\, s\end{aligned}$$

When we measure the input register $|\psi\rangle$ after Fourier sampling we will get a vector $|y\rangle$ such that $y \cdot s = 0 \bmod 2$. This will happen because the probability all ys which have an odd dot product will become 0.

Now we have a way to find some vector $|y\rangle$ such that,

$$y_n s_n \oplus y_{n-1} s_{n-1} \oplus \cdots \oplus y_1 s_1 = 0$$

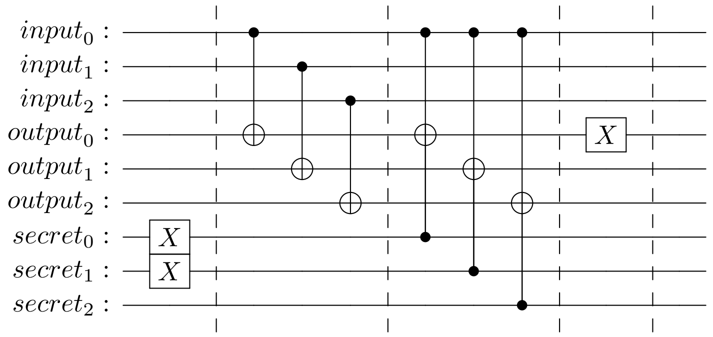
To find the secret bit string s we have to get $n-1$ independent linear equations as above. It can be shown that the probability of finding these $n-1$ equations in n tries is $\frac{1}{4}$. So if we perform the measurements $10n$ times the probability that we don't get independent equations is 0.05 which is very less. After getting the equations they can be solved to find the secret string s using Gaussian elimination, which takes $\mathcal{O}(n^3)$ time.

3 Implementation

3.1 Implementing the Oracle

There are many ways in which we can implement the oracle for Simon's Algorithm, here we look at one such implementation. Our basic goal is to create a 2-1 mapping from domain to range of $f(x)$. This can be realised by the following procedure -

- Encode the secret string using the X gate wherever there is a 1.
- Copy the input register to output register using $CNOT$ gates.
- We know that $s \neq 0^n$, so there exists at least one bit in s which has the value 1. Let us call this bit as the **MSB** bit. We now apply $CCNOT$ gate on output bit which is controlled by the corresponding secret bit and the input **MSB** bit. This creates a 2-1 mapping.
- Now we randomly flip some of the output qubits to further obfuscate the oracle. This step is optional and can be made even more complex so to make it harder to guess the secret string s.



Circuit for Simon's oracle for $s = 110$

Qiskit implementation

Following is the code for the Oracle using IBM's Qiskit library. The complete source along with the test function can be found in the Github repository.

Oracle Code

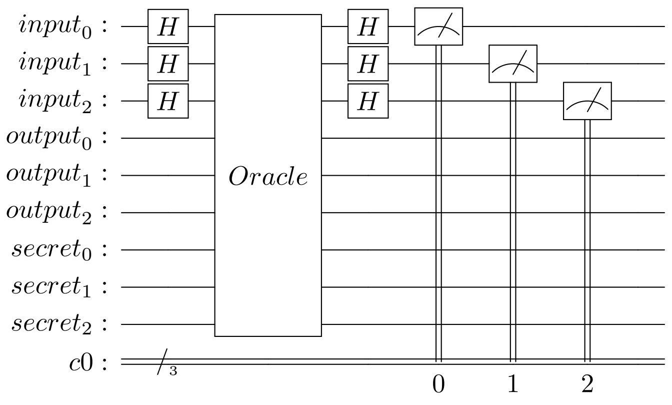
```
import itertools
import random
import time
from typing import List, Tuple

from qiskit import (Aer, ClassicalRegister,
                    QuantumCircuit, QuantumRegister,
                    execute)
from qiskit.circuit import Gate

random.seed(time.monotonic())


def simons_oracle(size: int, secret_input: List[int] = None):
    """ Returns a quantum gate which acts as
        an Oracle for Simon's Algorithm
    """
    # Register holding the secret string
    secret = QuantumRegister(size, "secret")

    # Input and Output registers
    input = QuantumRegister(size, "input")
    output = QuantumRegister(size, "output")

    # Initialize circuit
    oracle = QuantumCircuit(input, output, secret)

    # Generate random secret string
    if not secret_input:
        secret_input = [0 for i in range(size)]
        while secret_input.count(0) == size:
            secret_input = [
                random.choice([0, 1]) for i in range(size)
            ]

    # Encode it in the quantum register
    for i in range(size):
        if secret_input[i]:
            oracle.x(secret[i])
```

Oracle Code

```python
# Copy input register to output register
for i in range(size):
    oracle.cx(input[i], output[i])

# Find msb of secret string
msb = secret_input.index(1)

# Create 2-1 mapping
for i in range(size):
    oracle.ccx(input[msb], secret[i], output[i])

# Randomly flip qubits for further obfuscation
for i in range(size):
    if i % 3 == 0:
        oracle.x(output[i])

return oracle.to_gate(label="oracle"), secret_input
```

3.2 Implementing the algorithm

The implementation of actual Simon's algorithm is pretty straightforward. After we get the set of equations we need, we use Gaussian elimination to solve them and find the secret string.



Simon's Algorithm circuit for $n = 3$

Qiskit implementation

Following is the code for the the actual algorithm using IBM's Qiskit library. The complete source can be found in the Github repository.

Simon's Algorithm Code

```
import sys

from qiskit import (Aer, ClassicalRegister,
                    QuantumCircuit, QuantumRegister,
                    execute)
from qiskit.visualization import plot_histogram

from simons_oracle import simons_oracle


# Get backend
backend = Aer.get_backend('qasm_simulator')

# Length of input
n = None
try:
    n = int(sys.argv[1])
except IndexError:
    n = 4
except ValueError:
    print("Invalid size")
    sys.exit(1)

# Secret input (optional)
secret_input = None
try:
    secret_input = sys.argv[2]
    secret_input = [int(bit) for bit in secret_input]
    if len(secret_input) != n:
        print("Length of secret string and size "
              "of input should be the same")
        exit(1)
except IndexError:
    pass


# Initialise input, output and secret registers
input = QuantumRegister(n, "input")
output = QuantumRegister(n, "output")
secret = QuantumRegister(n, "secret")
result = ClassicalRegister(n)
```

Simon's Algorithm Code

```
(oracle, secret_input) = simons_oracle(n, secret_input)
print(f"Secret string - "
      f"{''.join([str(bit) for bit in secret_input])}")

# Initialize circuit
circuit = QuantumCircuit(input, output, secret, result)

# Actual circuit
circuit.h(input)
circuit.append(oracle, [*input, *output, *secret])
circuit.h(input)

# Perform measurement
circuit.measure(input, result)

print(circuit.draw())

# Take the top results and create a matrix out
# of it to solve the equations
res = execute(circuit, backend).result().get_counts()
res = sorted(res, key=lambda k: res[k])[:2**(n-1)]

# Convert to proper integer matrix
mat = [[int(bit) for bit in bit_string] for bit_string in res]
for row in mat:
    row.reverse()

# Perform Gaussian Elimination
x, y = len(mat), n
cur_row, cur_col = 0, 0
while cur_row < x and cur_col < y:
    if mat[cur_row][cur_col] == 0:
        non_zero_row = -1
        for j in range(cur_row+1, x):
            if mat[j][cur_col] == 1:
                non_zero_row = j
                break
        if non_zero_row == -1:
            cur_col += 1
            continue
        else:
            (mat[cur_col], mat[non_zero_row]) =
                (mat[non_zero_row], mat[cur_row])
```

Simon's Algorithm Code

```
    for j in range(cur_row+1, x):
        if mat[j][cur_col] == 1:
            for k in range(y):
                mat[j][k] = mat[j][k] ^ mat[cur_row][k]

    cur_row += 1

mat = list(filter(lambda row: row.count(0) < y, mat))

# Solve the reduced matrix
sol = [-1 for i in range(n)]
for i in range(n-1):
    if not mat[i][i]:
        sol[i] = 1
        for j in range(i+1, n):
            sol[j] = 0
        break

if sol.count(-1) == n:
    sol[-1] = 1

start = sol.index(1)-1
for i in range(start, -1, -1):
    cnt = 0
    for j in range(n):
        if sol[j] == 1 and mat[i][j] == 1:
            cnt += 1
    sol[i] = cnt % 2

print(f'Calculated Secret String: '
      f'{"".join([str(bit) for bit in sol])}')
```

4 Conclusion

Hence we have devised an algorithm which can solve a problem in polynomial time ($\mathcal{O}(n^3)$) with the help of quantum computers for which the best known solution is exponential time ($\mathcal{O}(2^n)$) using classical computer. This disproves the **Church-Turing Hypothesis** and shows the potential power of quantum computers.

Grover's Algorithm

Rahul Goel
2019111034

November 2020

Abstract

This document can be used as an introductory tutorial to Grover's Algorithm. It also includes certain peculiar observations that help to deepen the understanding about the differences between Quantum Computing and Classical Computing.

Contents

Note - It has been assumed that $N = 2^n$ throughout this sheet.

1 The Problem - Needle in a Haystack

The problem that Grover's Algorithm solves is the unstructured search problem. In simple words, among N given entities, our task is to find one unique entity that satisfies a particular property.
In most cases, the situation is random and the only way a classical computer knows how to solve it is by brute force. This can take $O(N/2)$ on average and up to $O(N)$ time in the worst case.
The problem of unstructured search is considered to be a hard problem for a classical computer as N gets bigger in size. Grover's Algorithm provides a quadratic speedup to this and reduces the number of queries (time taken) to $O(\sqrt{N})$. In fact, it has been proven that for a quantum computer, this is the best possible time complexity.
Note - The problem gets easier and the complexity gets reduced by a constant factor when the entities that satisfy the property are more in number.

2 How the Algorithm works

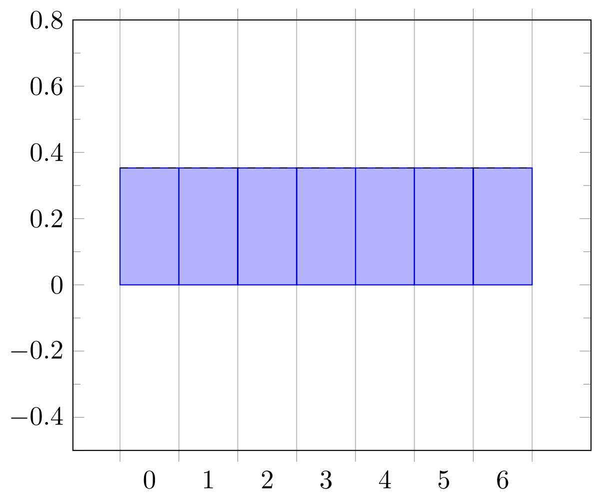
2.1 Intution 1

Initially, we start with n qubits in such a way that in their superposition, all the N states have equal amplitude (each has an equal probability of popping up when we measure the state).

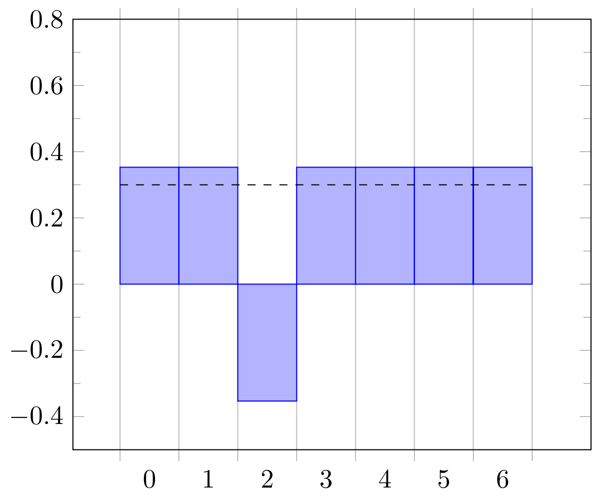
$$|\psi_{initial}\rangle = \sum_{0}^{N-1} \alpha_i |i\rangle \text{ where } \alpha_i = \frac{1}{\sqrt{N}}$$

Let $|j\rangle$ be our needle in the haystack. Now, by means of certain quantum gates, the amplitude α_j is first negated i.e. it becomes $-\alpha_j$. And then every single α_i is flipped about the mean of all amplitudes. As a result, α_j stands out more than every other α_i.

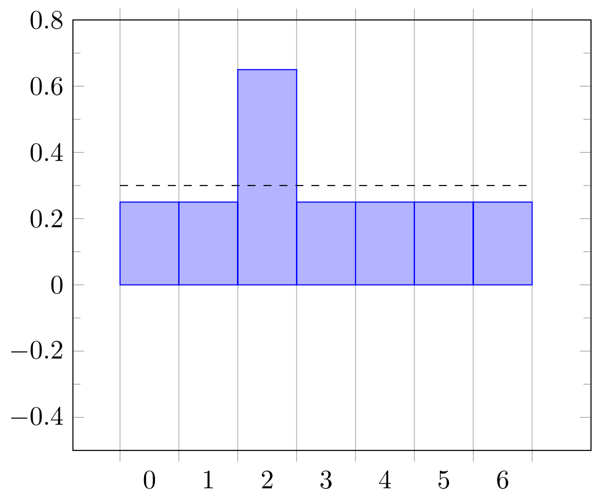
This has been explained by means of the following diagrams (Not to scale). Initially the amplitudes of all the states are equal and the sum of their squares is equal to 1.



Then a quantum circuit is applied to the state which flips the amplitude of the desired state.



Now the inversion of one of the amplitudes results in a slight decrement of the mean of all the amplitudes. If we now flip the amplitudes about their mean, we land up in the following state.



The amplitude of the needle in the haystack has increased and hence if we measure the state now, the chance of getting the correct value is more than it was previously. However, the sum of probabilities of getting other states is still greater than the probability of getting the answer.

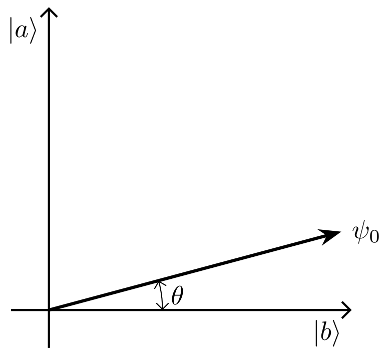
Hence, we repeat the negation and inversion steps a number of times so that when we measure the state, we can be pretty sure that the outcome is the answer (the desirable state).

Initially $\alpha_j = \frac{1}{\sqrt{N}}$. As we run iterations of the algorithm, α_j increases and α_i, $\forall\ i \neq j$, decreases, all in such a way that $\sum_0^{N-1} {\alpha_i}^2 = 1$. If we want the probability of getting $|j\rangle$ to be at least half, we need to pump up the magnitude α_j to $\frac{1}{\sqrt{2}}$. At the point in time when $\alpha_j = \frac{1}{\sqrt{2}}$, $\alpha_i \approx \frac{1}{\sqrt{2N}}\ \forall\ i \neq j$. Because the amplitude of the other states decreases continuously, we can say that throughout our iterations $\alpha_i \geq \frac{1}{\sqrt{2N}}$. Since in one iteration, the amplitude α_j increases by at least by $2\alpha_i$ $(i \neq j)$, the increment is at least $\sqrt{\frac{2}{N}}$. Therefore the number of iterations required is at least $\frac{1/\sqrt{2}}{\sqrt{2/N}} = \sqrt{N}$.
When the number of favourable states is M, the number of iterations comes out to be $\sqrt{\frac{N}{M}}$.

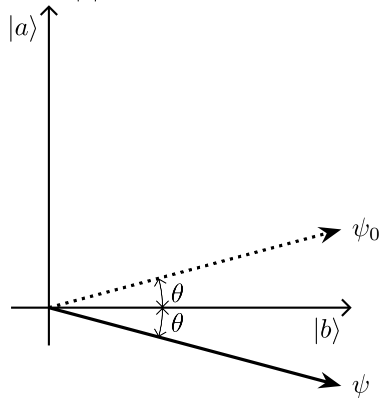
2.2 Intution 2

Let $|a\rangle$ represent our desirable state i.e. $|j\rangle$. And $|b\rangle$ represent the superposition of all the remaining states with equal amplitudes i.e. $|b\rangle = \sum_{i \neq j} \frac{1}{\sqrt{N-1}} |i\rangle$. It can be easily noticed that $|a\rangle$ and $|b\rangle$ are orthogonal to each other. Let ψ_0

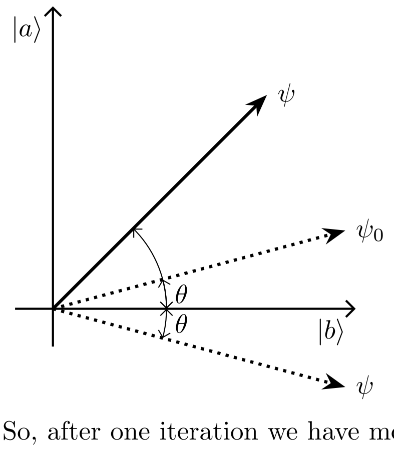
be our initial state which is $\sum_0^{N-1} \frac{1}{\sqrt{N}} |i\rangle$. Now initially $|\psi_0\rangle$ lies really close to $|b\rangle$ and it is the task of Grover's Algorithm to move it closer to $|a\rangle$.



The amplitude negation step leads to reflection of $|\psi_0\rangle$ about $|b\rangle$. This is true because $|b\rangle$ represents the superposition of all the undesirable states of equal amplitude. So the projection of $|\psi_0\rangle$ on $|b\rangle$ will remain as it is while the projection of $|\psi_0\rangle$ perpendicular to $|b\rangle$ will get negated. This is nothing but reflection about $|b\rangle$.



Now, we need to do inversion about the mean which is equivalent to reflection about the the equal amplitude state $\sum_0^{N-1} \frac{1}{\sqrt{N}} |i\rangle$.



So, after one iteration we have moved closer to the required state. We need to

carry these iterations certain number of times so that when we measure a state, it collapses to the desirable state with high probability.
Every time the inversion about mean is done using the equal amplitude state only. It happens to be our initial state too but this is true only for the first iteration and not for further iterations.

3 How to Implement the circuit?

Now that we know what to do, we have to question how to do it.

3.1 Negation of Amplitude

Our first task is to negate the amplitude of the desirable quantum state. The situation is like this that we have been given a quantum black box that outputs 1 for the desirable states and 0 for the undesirable states.
Insert diagram here.
In other words, given the superposition $|x\rangle|0\rangle$, the circuit spits out the superposition $|x\rangle|f(x)\rangle$. We know the quantum circuits are designed in such a way that the answer register gets XORed with the output $f(x)$. So if the input superposition is $|x\rangle|y\rangle$, then the output superposition will be $|x\rangle|y \oplus f(x)\rangle$.
With this knowledge, we can apply neat little hack and trick our black box into doing the task for us. We will initialise the answer register to $|-\rangle$. Since

$$\begin{aligned}
|-\rangle &= \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \\
|- \oplus f(x)\rangle &= \frac{1}{\sqrt{2}}\left(|0 \oplus f(x)\rangle - |0 \oplus f(x)\rangle\right) \\
|- \oplus f(x)\rangle &= \frac{1}{\sqrt{2}}\left(|f(x)\rangle - \left|\overline{f(x)}\right\rangle\right) \\
|- \oplus f(x)\rangle &= \frac{1}{\sqrt{2}}(-1)^{f(x)}\left(|0\rangle - |1\rangle\right) \\
|- \oplus f(x)\rangle &= (-1)^{f(x)}|-\rangle
\end{aligned}$$

Therefore, the entire transition will be

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

Now, since it is a superposition that is being looked at as a whole, the term $(-1)^{f(x)}$ can be associated with either $|x\rangle$ or $|-\rangle$. Putting $|x\rangle = \sum_0^{N-1} \alpha_i |i\rangle$, we get the output from the black box as $\sum_0^{N-1} \alpha_i |i\rangle (-1)^{f(x)}$. Now since $f(x) = 1$ for $|j\rangle$ and 0 for all others, the amplitude is negated only for $|j\rangle$. Thus we can do the amplitude negation easily.

3.2 Inversion about mean

Inversion about mean is the same as reflection about the uniform state i.e. reflection about the state $\sum_0^{N-1} \frac{1}{\sqrt{N}}$. Now for reflection about the uniform state, we can first map the uniform state to the all-zero state using a Hadamard Gate and then reflect about the all-zero state and then map the state back to the uniform state using another Hadamard Gate (Since Hadamard Gate is it's own inverse). So now we need a method to reflect about the all-zero state ($|00\ldots0\rangle$ or simply $|0\rangle$). Consider the following matrix :-

$$A = \begin{bmatrix} 1 & 0 & \ldots & 0 \\ 0 & -1 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & -1 \end{bmatrix}$$

This matrix maps negates all the states except for the state $|0\rangle$. This is the same as reflection about $|0\rangle$. This matrix is also a unitary matrix because $AA^\dagger = I$ i.e. the conjugate transpose is its inverse. So we can conclude that the sequence of matrices $H_N A H_N$ (where H_N stands for the unitary matrix corresponding to Hadamard Gate applied to n qubits.) does a reflection about the mean. And since all the matrices are unitary, they can be implemented using quantum gates.

So when we apply the operations of amplitude negation and inversion about mean repeatedly for around $\sqrt{N}$ times, the probability of getting the correct answer exceeds half and we are highly likely to get the current answer. We can perform this several times to be sure of the answer since the method is probabilistic.

4 Qiskit Code

Note - Please refer to the Jupyter Notebook for the output of the program and the explanatory comments and further experimentation.

Simulation Code

```
import matplotlib.pyplot as plt
import numpy as np

from qiskit import IBMQ, Aer, QuantumCircuit,
    ClassicalRegister, QuantumRegister, execute
from qiskit.providers.ibmq import least_busy
from qiskit.quantum_info import Statevector
from qiskit.visualization import plot_histogram
from qiskit.extensions import UnitaryGate
from math import sqrt

n = 5
N = 2 ** n

def initialize(qc, qubits):
    for q in qubits:
        qc.h(q)
    return qc

def get_oracle_matrix(N, values):
    oracle_matrix = np.identity(N)
    for value in values:
        oracle_matrix[value][value] = -1
    return oracle_matrix

def get_diffusion_matrix(N):
    diffusion_matrix = np.zeros((N, N), dtype = float)
    diffusion_matrix.fill(2 / N)
    diffusion_matrix -= np.identity(N)
    return diffusion_matrix
```

Simulation Code

```
oracle_matrix = get_oracle_matrix(N, values = [2])
oracle_unitary_gate = UnitaryGate(oracle_matrix)

diffusion_matrix = get_diffusion_matrix(N)
diffusion_unitary_gate = UnitaryGate(diffusion_matrix)

qc = QuantumCircuit(n)
qc = initialize(qc, [x for x in range(n)])

for i in range(5):
    qc.unitary(oracle_matrix, [x for x in range(n)])
    qc.unitary(diffusion_matrix, [x for x in range(n)])
qc.draw()

qc.measure_all()

qasm_simulator = Aer.get_backend('qasm_simulator')
shots = 1024
results = execute(qc, backend=qasm_simulator,
                    shots=shots).result()
answer = results.get_counts()
plot_histogram(answer, figsize = (15, 12))
```

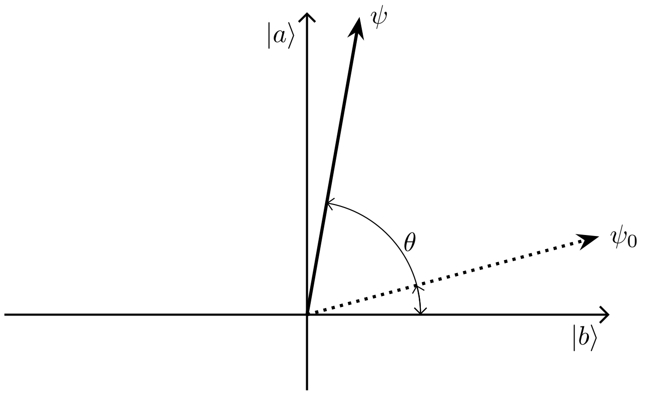
5 Findings

While it was mostly studying stuff discovered by others, I did observe some things on my own that most probably have already been observed.

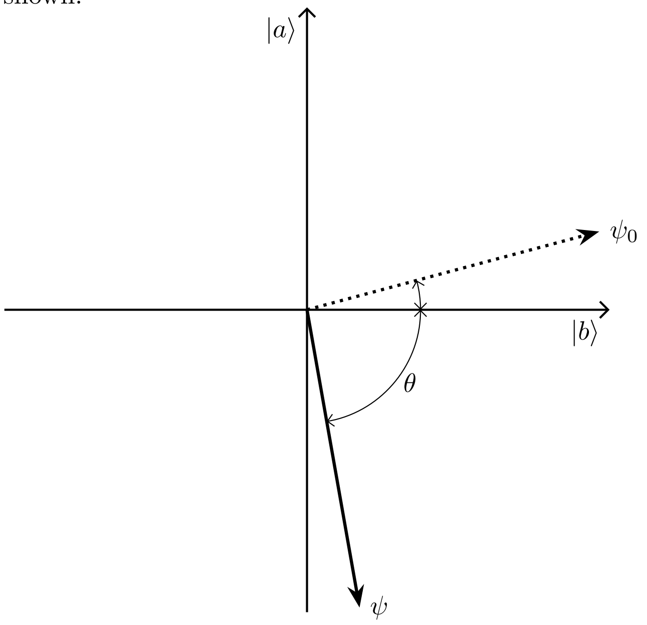
5.1 Going beyond $\sqrt{N}$ queries

In classical computers, it just doesn't hurt to go a few extra steps (since they don't harm the overall time complexity) just to be sure of our result. I tried this out on the implementation that I wrote and the results surprised me. After exceeding the required number of queries the amplitude of the desired state starts to decrease again and reaches a certain value and increases again and oscillates like this. Also, the period of this oscillation varies along with the number of qubits we are dealing with. When the number of qubits is small, the value decreases quite suddenly and even deviating by 1 query can alter the probability of the outcome easily. However, when the number of qubits is large this is smoothened out, and a query or two give or take doesn't affect the outcome by much.
All in all, minimum $\sqrt{N}$ queries are required and should be sufficient to give the answer. One should not go beyond this limit. Rather, one should re-conduct the experiment from the initial state to verify the answer.
So, what may be happening has been attempted to be explained with the following diagrams.

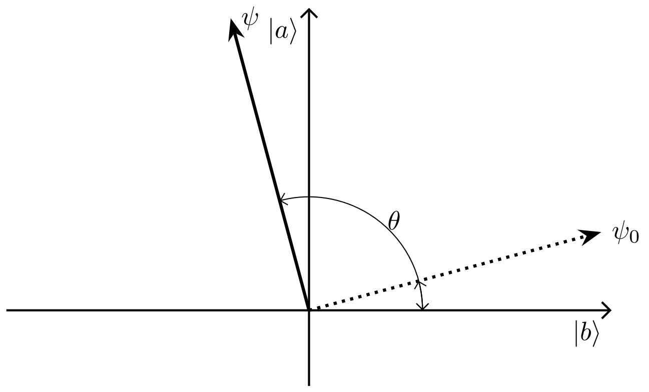
After $\sqrt{N}$ iterations, the state ψ is quite close to the desirable state. The probability of collapsing to the desirable state is as high as $80\% - 90\%$ (usual number as seen from the simulator code).



Now, when we do amplitude negation by reflecting about $|b\rangle$, we get state as shown.



And when we do the inversion about mean, we get the state as shown.



So exceeding one query does not make much difference, but when we do this a few number of times, ψ keeps on deviating from the required state. And then after some number of such iterations, it again starts to return to the position. And this "some number of iterations" is less than $\sqrt{N}$ because since the angle made from $|b\rangle$ is large, the iterations of the algorithm make huge changes in the states and thus requires fewer steps.
Note - This was observed when there was only one desirable state. With multiple desirable states a similar pattern was followed but not accurately.

5.2 Assumption about the given black box

It is assumed that a quantum circuit has already been implemented that identifies the answer that we use as a black box. Well, if such a black box has been implemented, the maker of the black box would already know about the desired state.
I think that maybe in future a situation may arise when the implemented black box is provided by nature and the algorithm may be used to find the desirable state.

6 Conclusion

Regardless of the black-box property mentioned above, this amazing algorithm does give an idea about the power of Quantum Computing. It gives a quadratic speed up in searching which can be used in database searching. The quadratic speed up can be used to solve some NP-Complete Problems by doing exhaustive searching over the set of all possible solutions (though not for large N as exponential term still dominates).