Grover's Search Algorithm

Indranil Ghosh
Jadavpur University, Physics Department
UG-III
Email: indranilg49@gmail.com

January 7, 2018

Abstract

This project is to compare the simulation of Grover's search algorithm for two qubit systems, between IBM's Quantum Experience's real quantum simulator and a Python program that has been designed, available in Github (https://github.com/indrag49/Quantum-SimuPy), for simulating the ideal case. Packages used: Numpy, Pandas and Mathplotlib. The quantum gates and the functions required for this simulation are available in the program and can be easily implemented. The moto of this project is to use python packages to design a quantum simulator for performing simple quantum computations.

1 Introduction

Grover's Algorithm, devised by *Lov Grover* in the year 1996, is a quantum search algorithm, that searches through a N-dimensional search space. Instead of searching for the desired element, the algorithm focuses its search on the index of the element, i, such that $i \in [0, N)$. The *amplitude amplification trick* is used as a core process here and the algorithm takes $\mathcal{O}(\sqrt{N})$ steps to achieve its goal compared to $\mathcal{O}$(N) steps taken by the classical counterpart.

2 The Problem

Suppose there is a database with N unsorted elements. We want to locate the index an element with a desired property, marked $\boldsymbol{w}$ from this database. We assume the value of N in such a way that the index can be stored in $n = \log_2 N$ bits, i.e., $N = 2^n$. This search problem has M solutions such that $1 <= M <= N$. For a classical search, $M = \frac{N}{2}$ on average and $M = N$ for the worst case scenario. But, on a quantum computer, M is roughly $\sqrt{N}$. Thse Grover's search algorithm provides a quadratic speedup, which is a time-saver for long lists, i.e for problems which may take million of years for the most powerful classical supercomputers in our possession, to solve.

3 What Is The Oracle?

The convinient way to encode the given list of N terms is by a function f such that $f(x) = \delta_{xw}$ where $(x, w) \in [0, N-1]$.

$$f(x) = 1, x = w$$
$$f(x) = 0, x \neq w$$

It follows $f(w) = 1$ for the winning index. At first we choose a binary encoding of the system , $x, w \in [0, 1]^n$ and represent the indices in terms of qubits in a quantum computer(x maps to 0 and w maps to 1). Next we define the *Oracle*, a quantum black box, U_f which is itself a unitary matrix, such that,

$$U_f |x\rangle = -|x\rangle \text{ for } x = w, f(x) = 1, \text{ x is a solution of the search}$$
$$U_f |x\rangle = |x\rangle \text{ for } x \neq w, f(x) = 0, \text{ x is not a solution of the search}$$

It is established that $U_f |x\rangle = (-1)^{f(x)} |x\rangle$.

The action of the *Oracle* can also be depicted as,

$$U_f |x\rangle |q\rangle = |x\rangle |q \bigoplus f(x)\rangle$$

Here, $|x\rangle$ is the index register, $\bigoplus$ denotes addition modulo 2 and the oracle qubit $|q\rangle$, a single qubit which flips if $f(x) = 1$ and else remains unchanged. It is seen that with N items with M solutions we need to apply the search oracle $\mathcal{O}(\sqrt{\frac{N}{M}})$ times to attain the solution.

4 A Deep Dive Into The Amplitude Amplification Technique

The algorithm uses single n qubit register and its goal is to find the solution of the search by applying the smallest number of oracles possible. In the beginning, the computer is in the state $|0\rangle^{\bigoplus n}$. Application of the Hadamard transform puts the computer in the state of uniform superposition $|s\rangle$,

$$|s\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{x=N-1} |x\rangle$$

Now, performing a measurement in the standard basis would result in the collapse of the superposition to any one of the basis states with the same probability of 2^{-n}. Hence, we need to try about 2^n steps on average to guess the correct result. The application of the Amplitude Amplification enhances the probability of the winning state to be located by the quantum computer. We consider two special states: the superposition state $|s\rangle$ and the winning state $|w\rangle$.

For the winning state, the oracle, U_f=U_w=$I - 2|w\rangle\langle w|$. We prove this from the following computations :

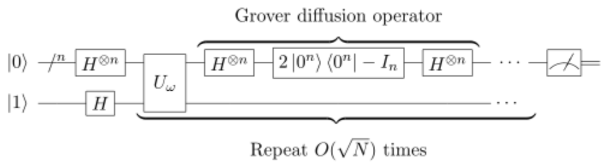
$$(I - 2|w\rangle\langle w|)|w\rangle = |w\rangle - 2|w\rangle\langle w|w\rangle = |w\rangle - 2|w\rangle = -|w\rangle = U_w|w\rangle, \text{ for } x = w$$
$$(I - 2|w\rangle\langle w|)|x\rangle = |x\rangle - 2|w\rangle\langle w|x\rangle = |x\rangle - 0 = |x\rangle = U_w|x\rangle, \text{ for } x \neq w$$

Now, we note the following computations:

$$1.\ \langle w|s\rangle = \frac{1}{\sqrt{N}}\sum_{x=0}^{N-1}\langle w|x\rangle = \frac{1}{\sqrt{N}} = \langle s|w\rangle$$
$$2.\ \langle s|s\rangle = \frac{1}{\sqrt{N}}\frac{1}{\sqrt{N}}\sum_{x=0}^{N-1}\sum_{x=0}^{N-1}\langle x|x\rangle = \frac{1}{N}N = 1$$

Besides the oracle, we also need another reflection operator, Grover's Diffusion operator, $U_s = 2\,|s\rangle\langle s| - I$.

5 Algorithm Procedure



Grover diffusion operator
$|0\rangle$
n
$H^{\otimes n}$
$|1\rangle$
H
U_ω
$H^{\otimes n}$
$2\,|0^n\rangle\langle 0^n| - I_n$
$H^{\otimes n}$
Repeat $O(\sqrt{N})$ times

Figure 1: Grover's Algorithm Circuit [Figure from Wikipedia]

We now state the Grover's iterator.After preparation of the superposition state we apply the oracle reflection U_w to the state, which corresponds the reflection of the state to $-|w\rangle$.

$$\begin{aligned} U_w\,|s\rangle &= (I - 2\,|w\rangle\langle w|)\,|s\rangle \\ &= I\,|s\rangle - 2\,|w\rangle\langle w|s\rangle \\ &= |s\rangle - \tfrac{2}{\sqrt{N}}\,|w\rangle \end{aligned}$$

This transformation means that the amplitude in front of the winning state becomes negative and the average amplitude decreases.

Now the additional reflection U_s is applied to the resultant state, $|s\rangle - \frac{2}{\sqrt{N}}|w\rangle$. We see, Grover's diffusion Operator, $U_s = H^{\otimes n}(2\,|0^n\rangle\langle 0^n| - I)H^{\otimes n} = 2(H^{\otimes n}|0^n\rangle\langle 0^n|H^{\otimes n}) - H^{\otimes n}H^{\otimes n} = 2\,|s\rangle\langle s| - I$, as stated above. Now,

$$\begin{aligned} U_s(|s\rangle - \tfrac{2}{\sqrt{N}}|w\rangle) &= (2\,|s\rangle\langle s| - I)(|s\rangle - \tfrac{2}{\sqrt{N}}|w\rangle) \\ &= 2\,|s\rangle\langle s|s\rangle - |s\rangle - \tfrac{4}{\sqrt{N}}|s\rangle\langle s|w\rangle + \tfrac{2}{\sqrt{N}}|w\rangle \\ &= 2\,|s\rangle - |s\rangle - \tfrac{4}{\sqrt{N}}\tfrac{1}{\sqrt{N}}|s\rangle + \tfrac{2}{\sqrt{N}}|w\rangle \\ &= |s\rangle - \tfrac{4}{\sqrt{N}}|s\rangle + \tfrac{2}{\sqrt{N}}|w\rangle \\ &= \tfrac{N-4}{N}|s\rangle + \tfrac{2}{\sqrt{N}}|w\rangle \end{aligned}$$

This two reflections corresponds to a rotation, that is rotates the initial state $|s\rangle$ towards the winner $|w\rangle$. Since the average amplitude was lowered by the first reflection, the second transformation boosts the negative amplitude of $|w\rangle$ to rougly three times its original value and lowers the other amplitudes.

We see that we need to apply the Grover's iterator roughly $\sqrt{N}$ times to concentrate on the winner. We note that in our simulations, we have considered $n = 2$, i.e $N = 4$. So $\frac{N-4}{N}|s\rangle + \frac{2}{\sqrt{N}}|w\rangle = |w\rangle$ for$N = 4$. The single application of the Grover's iterator returns us the winner index.

6 Comparisons for Circuits for $n = 2$, i.e, $N = 4$

As $N = 4$, there are 4 possible oracles U_f, one for each choice of the winner. The first part of the examples creates the superposition state $|s\rangle$, the second part applies the oracle U_f on $|s\rangle$ and the last part applies U_s on $U_f\,|s\rangle$.

6.1 A=00

6.1.1 IBM Quantum Experience results

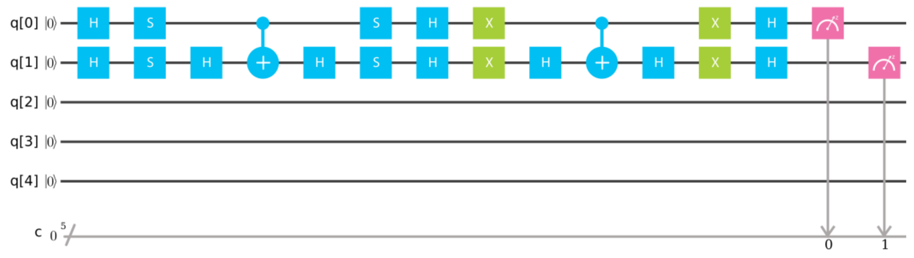


Figure 2: Algorithm Circuit[Figure from IBM Q experience's home page]

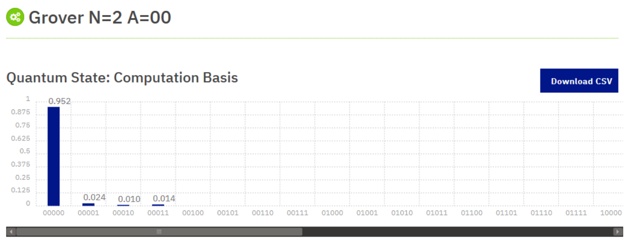


Figure 3: Probability Distribution, with noise[Figure from IBM Q experience's home page]

6.1.2 Python Simulation

```
k=np.kron
H=Hadamard(I2)
H_n=k(H, H)
h=k(I2, H)
# Uf
s=k(Phase(I2), Phase(I2))
s=h.dot(s)
s=CNOT(s)
s=h.dot(s)
Uf=k(Phase(I2), Phase(I2)).dot(s)
# Us
X_n=k(PauliX(I2), PauliX(I2))
t=h.dot(X_n.dot(H_n))
u=CNOT(t)
g=H_n.dot(X_n.dot(h))
Us=g.dot(u)
Gate=Us.dot(Uf.dot(H_n))
m=measure(Gate.dot(Q00))
print(m)
plot_measure(m)
```

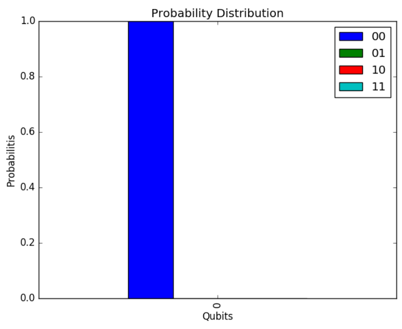


Figure 4: Probability Distribution, without noise

6.2 A=01

6.2.1 IBM Quantum Experience results

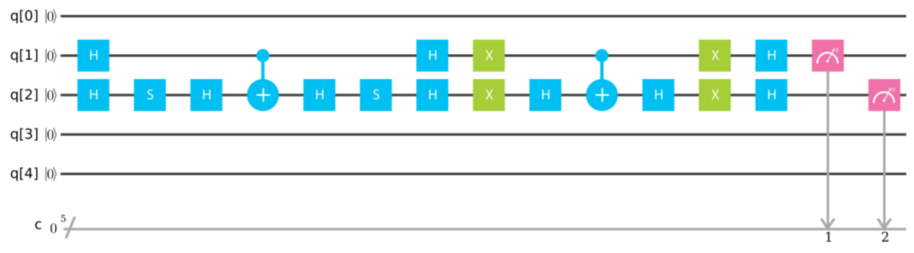


Figure 5: Algorithm Circuit[Figure from IBM Q experience's home page]

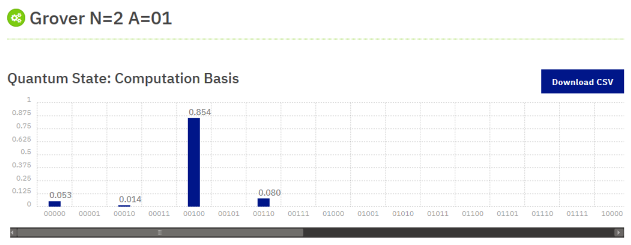


Figure 6: Probability Distribution, with noise[Figure from IBM Q experience's home page]

6.2.2 Python Simulation

```
k=np.kron
H=Hadamard(I2)
H_n=k(H, H)
h=k(I2, H)
# Uf
s=k(I2, Phase(I2))
s=h.dot(s)
```

```
s=CNOT(s)
s=h.dot(s)
Uf=k(I2, Phase(I2)).dot(s)
# Us
X_n=k(PauliX(I2), PauliX(I2))
t=h.dot(X_n.dot(H_n))
u=CNOT(t)
g=H_n.dot(X_n.dot(h))
Us=g.dot(u)
Gate=Us.dot(Uf.dot(H_n))
m=measure(Gate.dot(Q00))
print(m)
plot_measure(m)
```

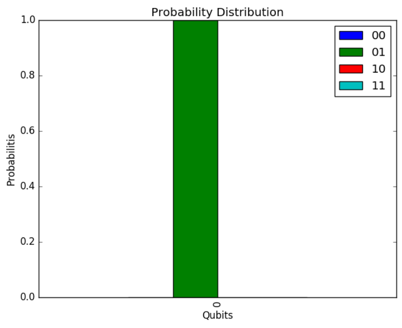


Figure 7: Probability Distribution, without noise

6.3 A=10

6.3.1 IBM Quantum Experience results

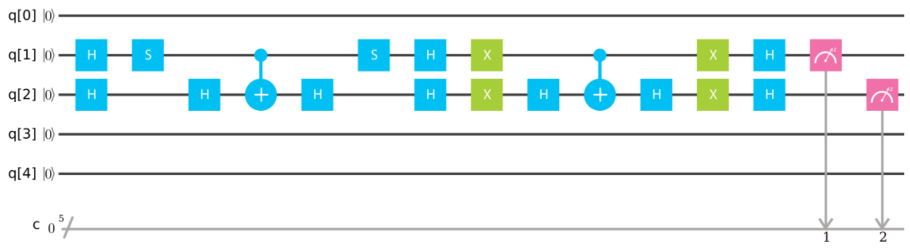


Figure 8: Algorithm Circuit[Figure from IBM Q experience's home page]

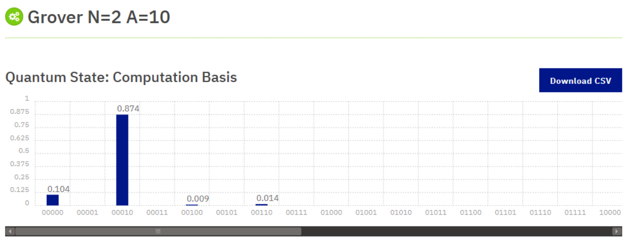


Figure 9: Probability Distribution, with noise[Figure from IBM Q experience's home page]

6.3.2 Python Simulation

```
k=np.kron
H=Hadamard(I2)
H_n=k(H, H)
h=k(I2, H)
# Uf
s=k(Phase(I2), I2)
s=h.dot(s)
s=CNOT(s)
s=h.dot(s)
Uf=k(Phase(I2), I2).dot(s)
# Us
X_n=k(PauliX(I2), PauliX(I2))
t=h.dot(X_n.dot(H_n))
u=CNOT(t)
g=H_n.dot(X_n.dot(h))
Us=g.dot(u)
G=Us.dot(Uf)
Gate=Us.dot(Uf.dot(H_n))
m=measure(Gate.dot(Q00))
print(m)
plot_measure(m)
```

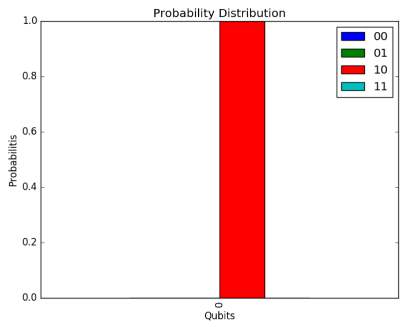


Figure 10: Probability Distribution, without noise

6.4 A=11

6.4.1 IBM Quantum Experience results

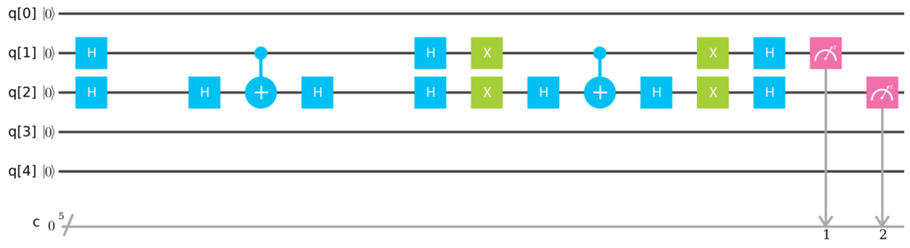


Figure 11: Algorithm Circuit[Figure from IBM Q experience's home page]

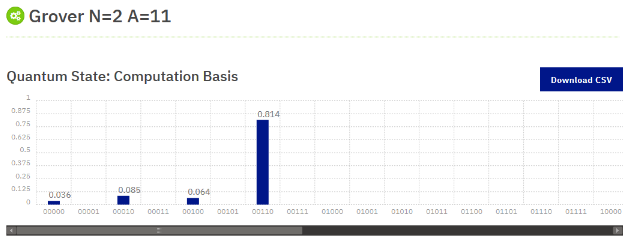


Figure 12: Probability Distribution, with noise[Figure from IBM Q experience's home page]

6.4.2 Python Simulation

```
k=np.kron
H=Hadamard(I2)
H_n=k(H, H)
h=k(I2, H)
# Uf
s=k(I2, I2)
s=h.dot(s)
s=CNOT(s)
s=h.dot(s)
Uf=k(I2, I2).dot(s)
# Us
X_n=k(PauliX(I2), PauliX(I2))
t=h.dot(X_n.dot(H_n))
u=CNOT(t)
g=H_n.dot(X_n.dot(h))
Us=g.dot(u)
G=Us.dot(Uf)
Gate=Us.dot(Uf.dot(H_n))
m=measure(Gate.dot(Q00))
print(m)
plot_measure(m)
```

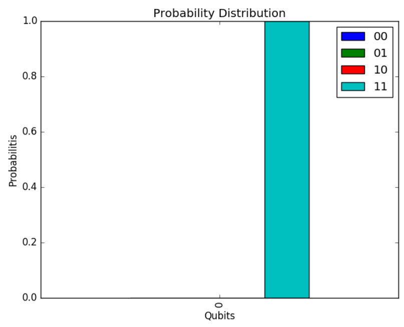


Figure 13: Probability Distribution, without noise

References

[1] `http://www-reynal.ensea.fr/docs/iq/QC10th.pdf`

[2] `https://en.wikipedia.org/wiki/Grover's algorithm`

[3] `https://quantumexperience.ng.bluemix.net/qx/experience`

[4] `https://github.com/indrag49/Quantum-SimuPy`