CS-308 : Calcul Quantique

Algorithme de Grover et 3-SAT

François Dumoncel et Souleyman Boudouh

June 3, 2022

Contents

※ Grover and 3-SAT problem

The 3-SAT problem is a well known instance of NP-complete problem class. It consists in having a list of classical bits $b_1, \ldots, b_n$ and finding the solution of a given predicate in the form of conjunctions of three-disjunctions. For instance, here is a 3-SAT predicate:

$$f(x, y, z) = (x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z) \wedge (x \vee \neg y \vee z) \tag{1}$$

One looks for solutions of $f(x, y, z) = 1$. One can check by exhaustive for this example that the solutions are 000, 100, 011, 101, 111. Using a Grover algorithm can help find solutions of a predicate with a quadratic speed-up over a naive exhaustive search. Hence we will implement it in this mini-project.

※ Question 1

2.1 Statement

We consider the 3-SAT predicate f defined as

$f(x,y,z) = (\bar{x} \vee \bar{y} \vee \bar{z}) \wedge (\bar{x} \vee \bar{y} \vee z) \wedge (\bar{x} \vee y \vee z) \wedge (x \vee \bar{y} \vee z) \wedge (x \vee y \vee \bar{z}) \wedge (x \vee y \vee z)$

Find all possible solutions.

2.2 Solution

An exhaustive search gives that

x	y	z	$f(x,y,z)$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1

and

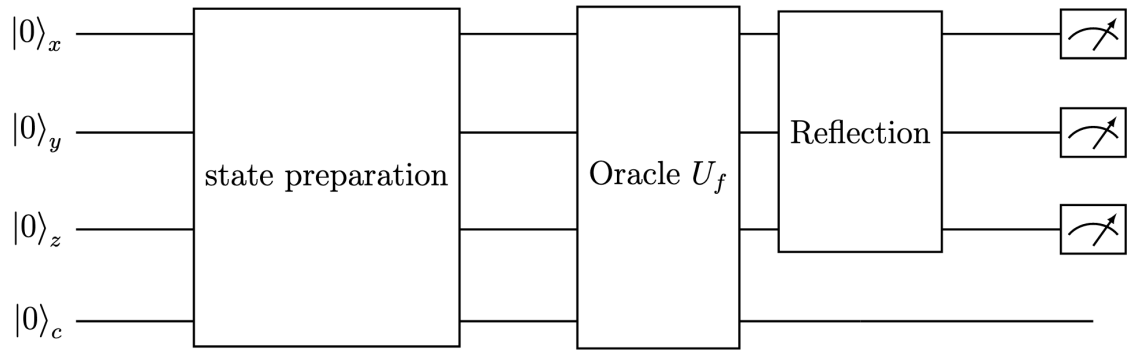
x	y	z	$f(x,y,z)$
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

and we can see that solutions are 011 and 101.

In the next questions you will implement Grover-style circuits to find solutions. How many times will you need to apply the Grover operator and why ?

Note that we have 2 among 8 possible input, this means that $N = 8$ and $M = \frac{N}{4} = 2$. We know that for this particular case it is enough to use the grover operator only once ($k = 1$).

We will now design a circuit to solve the above problem with



※ Question 2

3.1 Statement

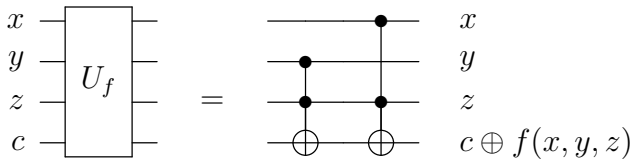
Design first the oracle gate U_f such that $U_f |xyzc\rangle = |xyz\rangle \otimes |c \oplus f(xyz)\rangle$

3.2 Solution

From de truth table above we can see that f can be simplified as

$$f(x, y, z) = (\neg x \wedge y \wedge z) \vee (x \wedge \neg y \wedge z)$$

which can be implemented as a cascade of two Toffoli gates



because

$$\begin{aligned} U_f \left|xyzc\right\rangle &= CCNOT_{yz}(c) CCNOT_{xz}(c) \left|xyzc\right\rangle \\ &= CCNOT_{yz}(c) \left|xyz\right\rangle \otimes \left|c \oplus (x \wedge z)\right\rangle \\ &= \left|xyz\right\rangle \otimes \left|c \oplus (x \wedge z) \oplus (y \wedge z)\right\rangle \end{aligned}$$

and thanks to the XOR operation in $(x \wedge z) \oplus (y \wedge z)$ we have perfectly implemented f's behaviour.

※ Question 3

4.1 Statement

Design the reflection operator and show the analytical proof of your design. You may need to find an appropriate extension of the reflection operator seen in the previous homework.

4.2 Solution

Using a result from the course, we have that our reflection operator is just

$$R = H^{\otimes 3}(\mathbb{I} - 2 \left|000\right\rangle \left\langle 000\right|) H^{\otimes 3}.$$

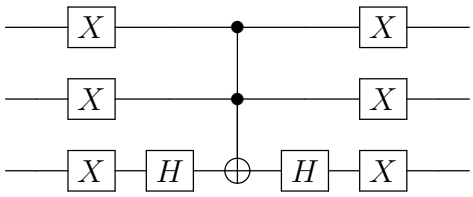
From this we can extract the phase conditional operator S

$$S = \mathbb{I} - 2 \left|000\right\rangle \left\langle 000\right|$$

Because, as seen course we have

$$\begin{aligned} S \left|b_1 b_2 b_3\right\rangle &= \left|b_1 b_2 b_3\right\rangle - 2 \left|000\right\rangle \left\langle 000 | b_1 b_2 b_3\right\rangle \\ &= \begin{cases} -\left|000\right\rangle & \text{if } b_1 b_2 b_3 = 000 \\ \left|b_1 b_2 b_3\right\rangle & \text{otherwise.} \end{cases} \end{aligned}$$

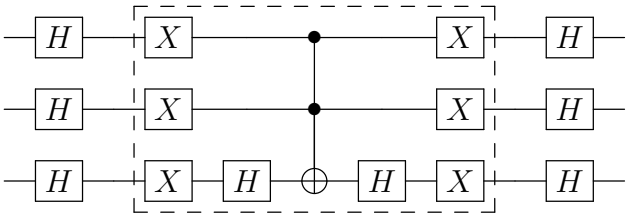
Using the homework 9, we see that a possible design for S is



and we prove that this circuit works. If the input is $|000\rangle$ then we have

$$\begin{aligned}
S|000\rangle &= (X^{\otimes 3})(\mathbb{I}\otimes\mathbb{I}\otimes H)CCNOT_{12}(3)(\mathbb{I}\otimes\mathbb{I}\otimes H)|111\rangle \\
&= (X^{\otimes 3})(\mathbb{I}\otimes\mathbb{I}\otimes H)CCNOT_{12}(3)|11\rangle\otimes\frac{|0\rangle-|1\rangle}{\sqrt{2}} \\
&= (X^{\otimes 3})(\mathbb{I}\otimes\mathbb{I}\otimes H)\frac{|111\rangle}{\sqrt{2}}-\frac{|110\rangle}{\sqrt{2}} \\
&= \frac{1}{\sqrt{2}}(X^{\otimes 3})\left(|11\rangle\otimes\frac{|0\rangle-|1\rangle}{\sqrt{2}}-|11\rangle\otimes\frac{|0\rangle+|1\rangle}{\sqrt{2}}\right) \\
&= \frac{1}{2}(X^{\otimes 3})\left(|110\rangle-|111\rangle-|110\rangle-|111\rangle\right) \\
&= -X^{\otimes 3}|111\rangle = -|000\rangle
\end{aligned}$$

as desired. We can also verify that for all other input $|\Psi\rangle \neq |000\rangle$ we have $S|\Psi\rangle = |\Psi\rangle$. Then our final reflection operator is (with the dashed box being the phase conditional operator)



※ Question 4

5.1 Statement

Run your circuit in IBMQ. Can you retrieve the solutions? Do you get any noise ?

5.2 Solution

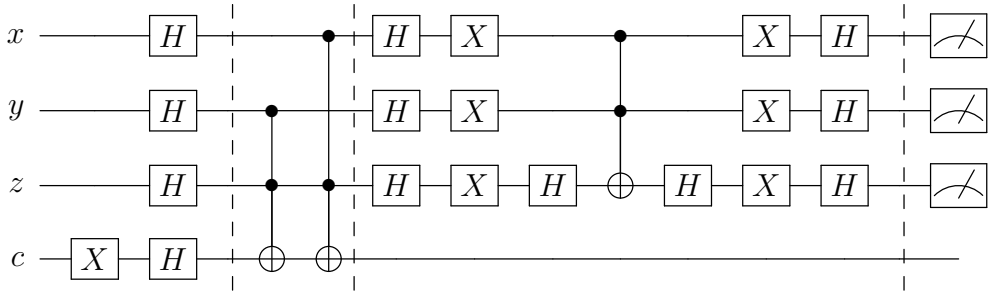
First, we initialize our Notebook with classical imports

```
# initialization
import numpy as np
from qiskit import *
from qiskit import QuantumCircuit
from qiskit.providers.ibmq import least_busy

# import basic plot tools
from qiskit.visualization import plot_histogram

#Loading IBMQ account
provider = IBMQ.load_account()
```

We then build the circuit

```
n = 4
qc = QuantumCircuit(n, n - 1)

qc.x(3)
qc.barrier()
qc.h(range(0, 4))
qc.barrier()

### Oracle
qc.toffoli(0, 2, 3)
qc.toffoli(1, 2, 3)
qc.barrier()

### Reflection
qc.h(range(0, 3))
qc.barrier()
qc.x(range(0, 3))
qc.h(2)
qc.toffoli(0, 1, 2)
qc.h(2)
qc.x(range(0, 3))
qc.barrier()
qc.h(range(0, 3))

qc.measure(np.arange(0, 3), np.arange(0, 3))

qc.draw(output='mpl')
```

and we obtain that our final circuit is



We run this on a simulator and we obtain the following result

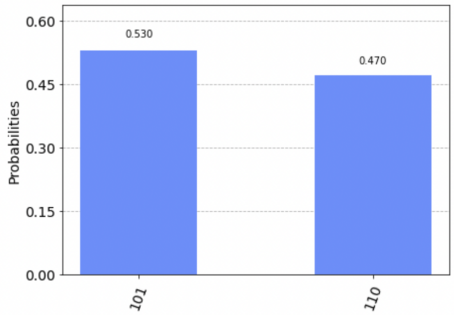


Figure 1: Simulator results

```
# Simulator
aer_sim = Aer.get_backend('aer_simulator')
result = aer_sim.run(assemble(qc)).result()
answer = result.get_counts()
plot_histogram(answer)
```

Figure 2: Code for running on a simulator

We can clearly see that our circuit gives the correct (and expected) result. But now the interesting part is to run this circuit on a real quantum device to see what is going on :

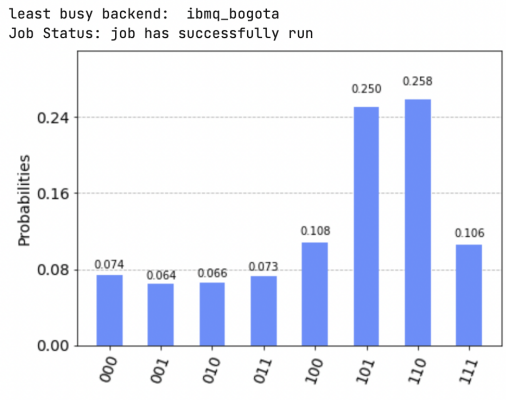


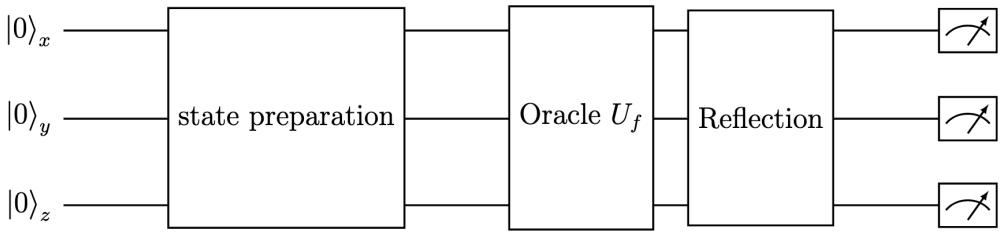
Figure 3: Real device results

```
# Real devices
backend = least_busy(provider.backends(filters=lambda x: 5 >= x.configuration().n_qubits>=2
                                       and not x.configuration().simulator
                                       and x.status().operational == True))

print ("least busy backend: ", backend)

from qiskit.tools.monitor import job_monitor
shots = 8192

transpiled_bv_circuit = transpile(qc, backend)
job = backend.run(transpiled_bv_circuit, shots=shots)
job_monitor(job, interval=2)

# Get the results from the computation
results = job.result()
answer = results.get_counts()
plot_histogram(answer)
```

Figure 4: Code for running on a real device

Surprisingly, we did not only observe the correct answers but also other bitstrings that are not solution. This is in fact *noise*, because on a NISQ device we know that quantum processors are very sensitive to the environment and may lose their quantum state due to quantum decoherence.

Let's assume that we can only use 3 qubits on the quantum machines. The circuit can in fact be made simpler in the following form :



※ Question 5

6.1 Statement

Find a block U_f such that $U_f |xyz\rangle = (-1)^{f(x,y,z)} |xyz\rangle$ and run again your circuit on the IBMQ machines. Compare your results with the previous question. Hint: In the above circuits you may use Toffoli gate.

6.2 Solution

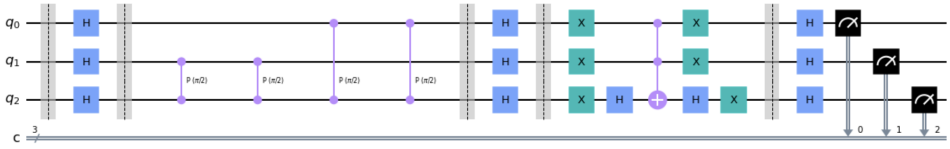
Note that we need to build an oracle that acts as

$$U_f |xyz\rangle = (-1)^{f(x,y,z)} |xyz\rangle = (-1)^{(y\wedge z)\vee(x\wedge z)} |xyz\rangle = e^{i\pi yz} e^{i\pi xz} |xyz\rangle$$

meaning that we just need to adjust the phase of the solution states to detect them. We achieved this by using four $\frac{\pi}{2}$-Control-Phase gate that puts a phase of $\frac{\pi}{2}$ behing the corresponding qubit if and only if $x = 1, y = 0$ and $z = 1$ or $x = 0, y = 1$ and $z = 1$. The code used for the simulation is presented below

```
from numpy import pi

n = 3
qc = QuantumCircuit(n, n)

qc.barrier()
qc.h(range(0,3))
qc.barrier()

### Oracle
qc.cp(pi/2, 1, 2)
qc.cp(pi/2, 2, 1)
qc.cp(pi/2, 2, 0)
qc.cp(pi/2, 0, 2)

qc.barrier()

### Reflection
qc.h(range(0,3))
qc.barrier()
qc.x(range(0,3))
qc.h(2)
qc.toffoli(0,1,2)
qc.h(2)
qc.x(range(0,3))
qc.barrier()
qc.h(range(0,3))

qc.measure(np.arange(0,3), np.arange(0,3))


qc.draw(output='mpl')
```

Hence, the new circuit is



Running directly on a real device, we obtain the following result

```
least busy backend:  ibmq_manila
Job Status: job has successfully run
```

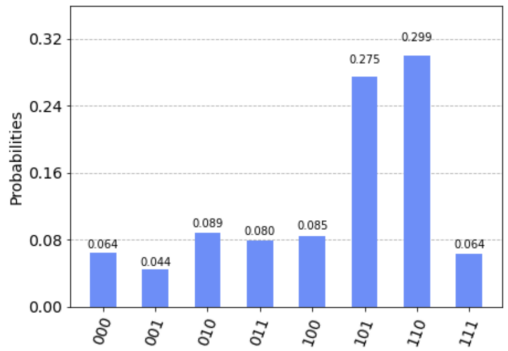


Figure 5: Real device results

```python
# Real devices
backend = least_busy(provider.backends(filters=lambda x: 5 >= x.configuration().n_qubits>=2
                                       and not x.configuration().simulator
                                       and x.status().operational == True))

print ("least busy backend: ", backend)

from qiskit.tools.monitor import job_monitor
shots = 8192

transpiled_bv_circuit = transpile(qc, backend)
job = backend.run(transpiled_bv_circuit, shots=shots)
job_monitor(job, interval=2)

# Get the results from the computation
results = job.result()
answer = results.get_counts()
plot_histogram(answer)
```

Figure 6: Code for running on a real device

We can observe pretty much the same result as before when we have used an ancilla bits. We have increase performance using three qubits instead of four but the price is that we have used two more gates in the oracle.