Grover_algorithm_qiskit

November 7, 2024

1 Quantum information and quantum computing - Problem set 08

1.0.1 *Problem 1* : Code Grover's search algorithm

In this notebook we are going to see a simple implementation of the Grover algorithm on a database of dimension $N = 8$ possibile data, so using $n = 3$ qubits.

Between all the possibile state $|xyz\rangle$, let's consider $|110\rangle$ and $|101\rangle$ as the solutions of our problem.

[1]:
```
#initialization
import matplotlib.pyplot as plt
import numpy as np
import math

# importing Qiskit
from qiskit_aer import QasmSimulator
from qiskit import QuantumCircuit, ClassicalRegister, QuantumRegister
```

Creating the oracle The first thing to make is a oracle function, that adds a phase to the states that are solution of our search problems. Considering $f(x) = 1$ iff x is a solution of the problem, the unitary has the form

$$U|x\rangle_n = (-1)^{f(x)}|x\rangle_n \tag{1}$$

In our case the oracle function is easy to make: we have to control if the first qubit (the last, for qiskit order) is 1 and then perform a X gate on second and third qubit.

Note that, since we already know the solutions, we can construct the oracle without the ancilla qubit.

[2]:
```
def oracle(circuit):
    circuit.cz(2,0)
    circuit.cz(2,1)

    circuit.barrier() # Barriers are added to divide the different parts of the␣
 ↪circuits
    return circuit
```

Creating the Grover gate Now we want to add the Grover gate. Remember that it is made of three parts:

- Apply Hadamard gates on all qubits
- Apply a phase shift to all the $|x\rangle_n$ except $|0\rangle_n$
- Apply again Hadamard to all the qubits

[3]:
```
def grover_gate(circuit):
    # Hadamard
    for i in range(circuit.num_qubits):
        circuit.h(i)

    # To add a phase at all the solutions except |0>, we create a set of
    # transformations that only |0> won't trigger
    circuit.z(0) # address |**1> states
    circuit.x(0) # now target |**0> states
    circuit.cz(0,1) # for |*10> states
    circuit.cx(1,2) # control if the second qubit is in 1
    circuit.cz(0,2) # for |100> state

    # now bring the qubit back to their original state
    circuit.cx(1,2)
    circuit.x(0)


    #Hadamard again
    for i in range(circuit.num_qubits):
        circuit.h(i)

    circuit.barrier()
    return circuit
```

Construct the circuit We can now assemble the Grover circuit for our problem. As a first thing, apply the Hadamard:

[4]:
```
grover= QuantumCircuit(3,3)

for i in range(3):
    grover.h(i)

grover.barrier()
```

[4]:
```
CircuitInstruction(operation=Instruction(name='barrier', num_qubits=3,
num_clbits=0, params=[]), qubits=(Qubit(QuantumRegister(3, 'q'), 0),
Qubit(QuantumRegister(3, 'q'), 1), Qubit(QuantumRegister(3, 'q'), 2)),
clbits=())
```

Then the oracle and the grover gate (since we have 2 solutions in $N = 8$ possibilities, one iteration

is sufficient to obtain the *exact* result):

[5]:
```
grover = oracle(grover)
grover = grover_gate(grover)
```

Don't forget to measure!

[6]:
```
for i in range(grover.num_qubits):
    grover.measure(i,i)
```

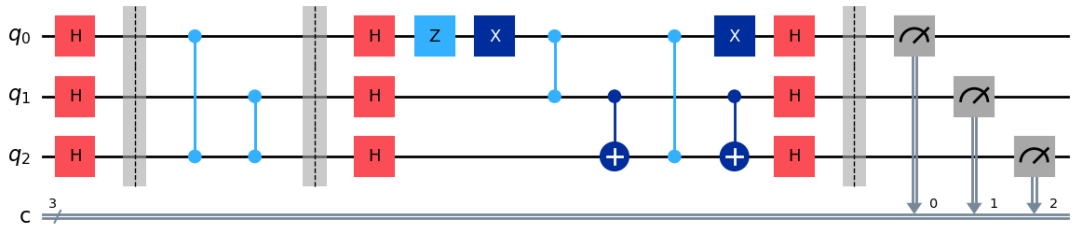
Print the circuit

[7]:
```
# We can simply print the circuit
#print(grover)

# Or draw it just like in IBM Quantum Experience
# to do this, install pylatexenc with 'pip install pylatexenc'
# and then use the draw function

grover.draw('mpl')
```

[7]:



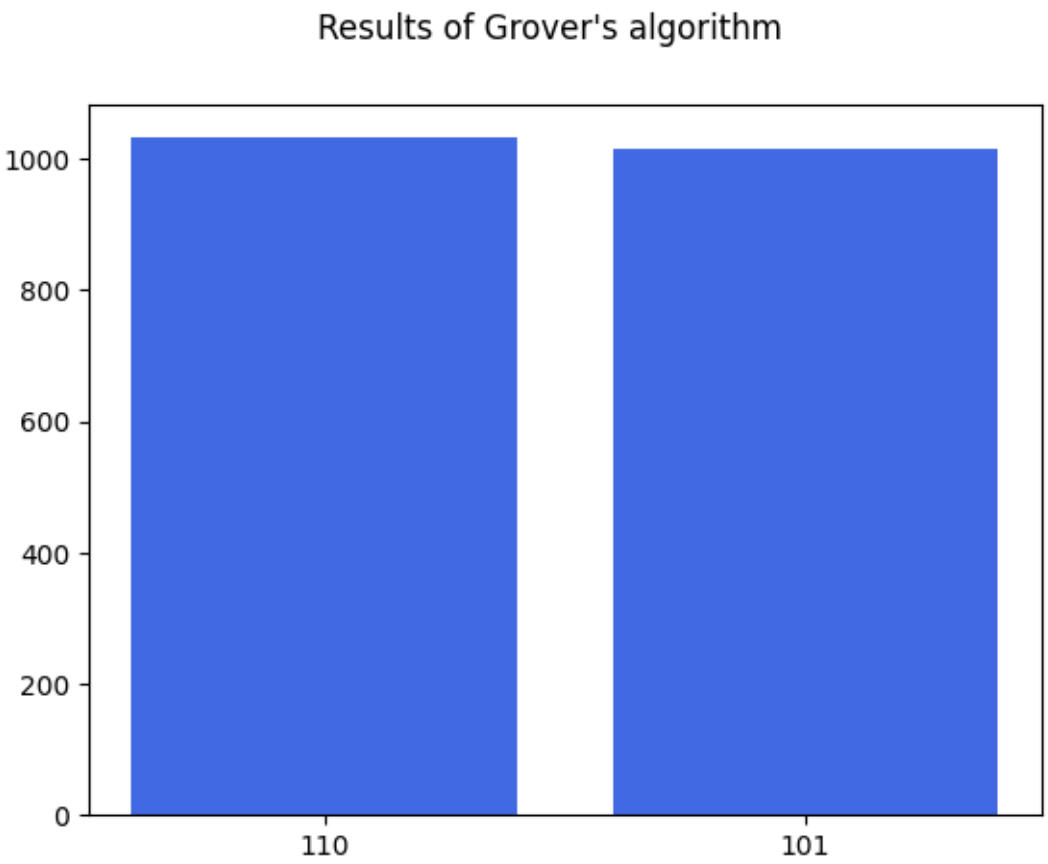
Run the algorithm on QASM

[8]:
```
backend = QasmSimulator()
results = backend.run(grover, backend=backend, shots=2048).result()
answer = results.get_counts()
```

Visualize the results

[9]:
```
plt.suptitle("Results of Grover's algorithm")
plt.bar(answer.keys(), answer.values(), color='royalblue')
plt.show()

print(answer)
```




```
{'110': 1032, '101': 1016}
```

As you can see, the results are exactly what we expected! Since we prepared a equiprobable superposition of the two result states, half of the time we get $|101\rangle$ and the other half $|110\rangle$.

In this case we have only the solutions of the problem as outcome of the measurements, because with one rotation you have exactly the superposition of the solutions.

In general, the aim of the Grover's algorithm is to get as close as possibile to the superposition of the solutions, in order to have statistically the right answers.

1.0.2 *Extra* : Run with noise model

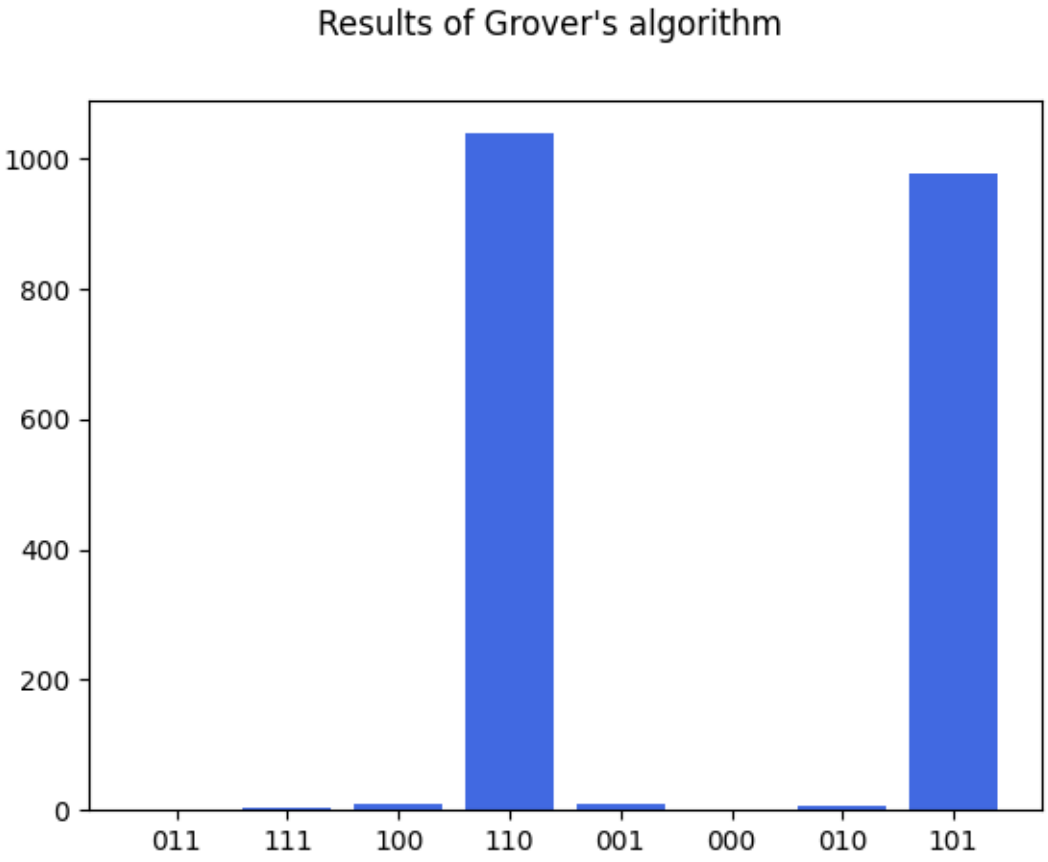
[10]:
```
from qiskit_aer.noise import NoiseModel
from qiskit.providers.fake_provider import GenericBackendV2

fake_backend = GenericBackendV2(num_qubits=3)
noise_model = NoiseModel.from_backend(fake_backend)
shots = 2048
```

```
results = fake_backend.run(grover, backend=fake_backend,␣
  ↪shots=shots,noise_model = noise_model).result()
answer = results.get_counts()
```

[11]:
```
plt.suptitle("Results of Grover's algorithm")
plt.bar(answer.keys(), answer.values(), color='royalblue')
plt.show()

print(answer)
```




```
{'011': 1, '111': 5, '100': 9, '110': 1037, '001': 10, '000': 2, '010': 7,
'101': 977}
```

[ ]: