QCQT-QY3 Quantum Algorithms and Quantum Information 2024-2025
Professor Karafyllidis Ioannis
Topalidis Dimitrios
Assignment 2

The Grover's quantum algorithm searches an unstructured database containing N elements to find X_i.
Each element is numbered from 0 to $N-1$. We also have a system that can recognize whether an element is the one we are searching for or not is provided, which we call the Oracle.
The Oracle that we will use for this assignment is:

$$f(x) = 1 \text{ if } x = x_i \text{ and } f(x) = 0 \text{ if } x \neq x_i$$

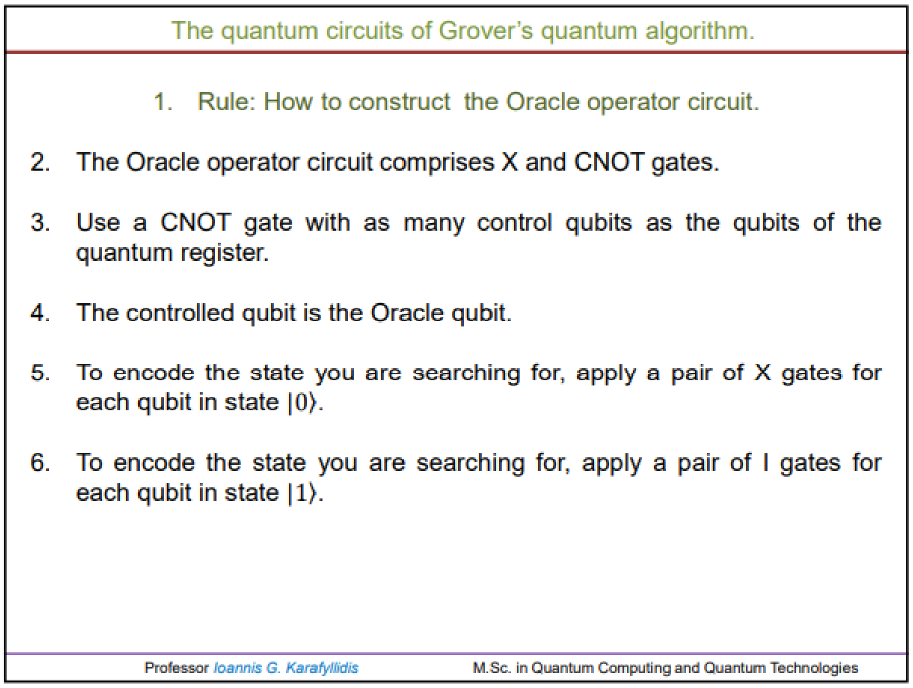
The algorithm provides a quadratic improvement over the best classical algorithm.

For this assignment, we are asked to use Grover's quantum algorithm to find the state

$$|x_i\rangle = |1001\rangle$$

My implementation takes into account the general case and can work for any given state. However the oracle used is the one presented in the lecture notes.
In our example, we will test the implementation with the state $|x_i\rangle = |1001\rangle$ as required by the exercise

The Grover's algorithm makes use of two operators:
The oracle operator, which is implemented with the **oracle_operator(s)** method. This method takes as input the state we are searching **s** and returns the respective circuit of the oracle operator. The operator is implemented according to the instructions given in the lecture notes, which are shown in the image below.



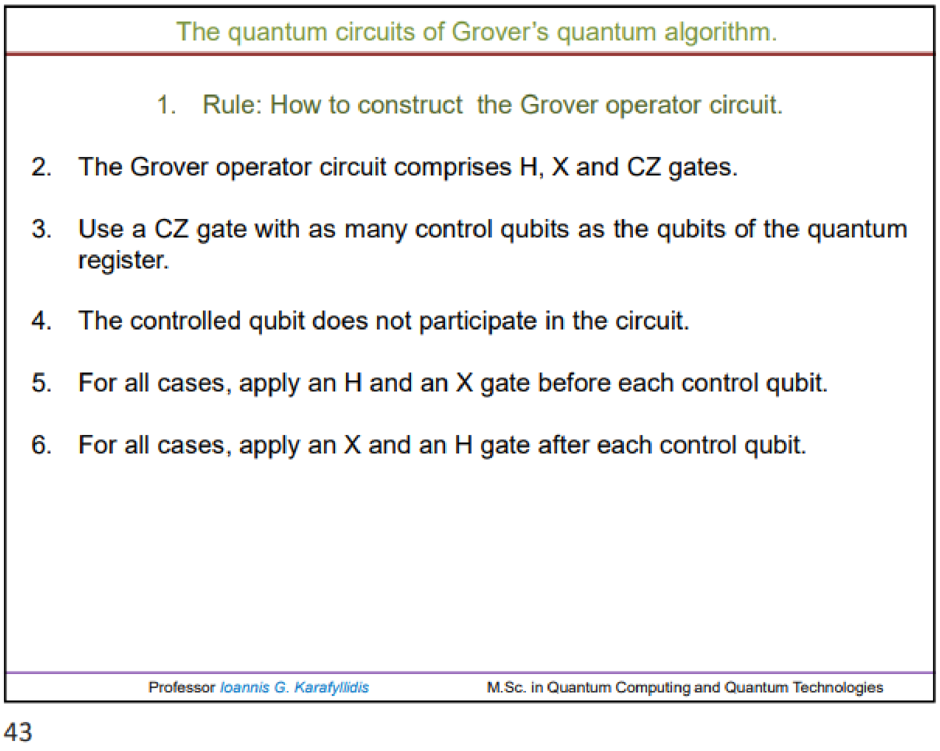
The quantum circuits of Grover's quantum algorithm.

1. Rule: How to construct the Oracle operator circuit.
2. The Oracle operator circuit comprises X and CNOT gates.
3. Use a CNOT gate with as many control qubits as the qubits of the quantum register.
4. The controlled qubit is the Oracle qubit.
5. To encode the state you are searching for, apply a pair of X gates for each qubit in state |0⟩.
6. To encode the state you are searching for, apply a pair of I gates for each qubit in state |1⟩.

Professor Ioannis G. Karafyllidis M.Sc. in Quantum Computing and Quantum Technologies

37

The Grover's diffusion operator, which is implemented with the **diffusion_operator(n)** method. This method takes as input the length **n** of the state we are searching for(or the number of qubits) and returns the respective circuit of the difussion operator. The operator is implemented according to the instructions given in the lecture notes, which are shown in the image below.



The quantum circuits of Grover's quantum algorithm.

1. Rule: How to construct the Grover operator circuit.
2. The Grover operator circuit comprises H, X and CZ gates.
3. Use a CZ gate with as many control qubits as the qubits of the quantum register.
4. The controlled qubit does not participate in the circuit.
5. For all cases, apply an H and an X gate before each control qubit.
6. For all cases, apply an X and an H gate after each control qubit.

Professor *Ioannis G. Karafyllidis* M.Sc. in Quantum Computing and Quantum Technologies

43

```python
def oracle_operator(s):

    n = len(s)

    qc = QuantumCircuit(n+1)

    for i, bit in enumerate(s):
        if bit == '0':
            qc.x(i)

    qc.mcx(list(range(n)), n)

    for i, bit in enumerate(s):
        if bit == '0':
            qc.x(i)

    return qc


def diffusion_operator(n):

    qc = QuantumCircuit(n)

    qc.h(list(range(n)))
    qc.x(list(range(n)))
    qc.h(n-1)
    qc.mcx(list(range(n-1)), n-1)
    qc.h(n-1)
    qc.x(list(range(n)))
    qc.h(list(range(n)))

    return qc
```

Having defined those two required operators, we implement the Grover's algorithm as a sequential application of those operators approximately $\lceil \frac{\pi}{4}\sqrt{N} \rceil - 1$ times, with $N = 2^n$, where **n** is the length of the state we are searching for.
The main implementation is found in the method **grovers_algorithm(oracle, n)**
This method takes two arguments:

- **oracle**, which is a Qiskit QuantumCircuit that implements the oracle for the state we are searching
- **n**, which is the length of the state we are searching for

Then, we have the following steps:

1. We initialize a quantum circuit with $\mathbf{n+1}$ qubits and **n** classical bits The first n qubits represent the state and the last one is the control/ ancilla qubit.

```
        qc = QuantumCircuit(n+1, n)
```

2. We apply an Pauli X gate to the ancilla qubit, in order to prepate its state to $|1\rangle$, and then apply a Hadamard gate separately.

```
        qc.x(n)
        qc.h(n)
```

3. We apply a Hadamard gate to all qubits except from the ancilla qubit.

```
        qc.h(list(range(n)))
```

4. We obtain the number of times to apply the oracle and diffusion operators.

```
        N = int(np.ceil(np.pi / 4 * np.sqrt(2**n))) -1
```

5. We apply iteratively the oracle operator and the diffusion operator to the quantum circuit $\lceil \frac{\pi}{4}\sqrt{N} \rceil - 1$ times

```
        for _ in range(N):
            qc.barrier()
            qc.compose(oracle, range(n+1), inplace=True)
            qc.barrier()
            qc.compose(diffusion_operator(n), range(n), inplace=True)
```

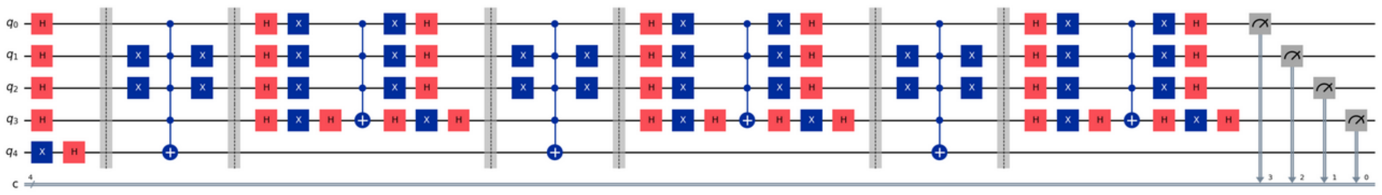
6. We measure all qubits except of the ancilla qubit, in the respective quantum register

```
        qc.measure(list(range(n)) , reversed(list(range(n))))
```

```
    def grovers_algorithm(oracle, n):

        #Step 1
        qc = QuantumCircuit(n+1, n)

        #Step 2
        qc.x(n)
        qc.h(n)


        #Step 3
        qc.h(list(range(n)))

        #Step 4
        N = int(np.ceil(np.pi / 4 * np.sqrt(2**n))) -1

        #Step 5
        for _ in range(N):
            qc.barrier()
            qc.compose(oracle, range(n+1), inplace=True)
            qc.barrier()
            qc.compose(diffusion_operator(n), range(n), inplace=True)

        #Step 6
        qc.measure(list(range(n)) , reversed(list(range(n))))

        return qc
```

We now construct the required oracle by passing **"1001"** as a parameter, which is the state that the we are asked to search. After that we "build" our circuit, by calling the **grovers_algorithm** function and passing the obtained oracle and 4 as parameters. We also draw it, to have a nice visual representation.

```
    #Example required by the assignment
    oracle = oracle_operator("1010")
    qc = grovers_algorithm(oracle, 4)
    qc.draw('mpl')
```



We then proceed to transpile our quantum circuit and simulate it **without noise**, using the **StatevectorSampler** and to plot the results.

```
    #Simulation using StatevectorSampler
    pm = generate_preset_pass_manager(optimization_level=3)
    transpiled_qc = pm.run(qc)
    statevectorSampler = StatevectorSampler()
    job = statevectorSampler.run([transpiled_qc])
    pub_result = job.result()[0]
    counts = pub_result.data.c.get_counts()
    plot_histogram(counts)
```

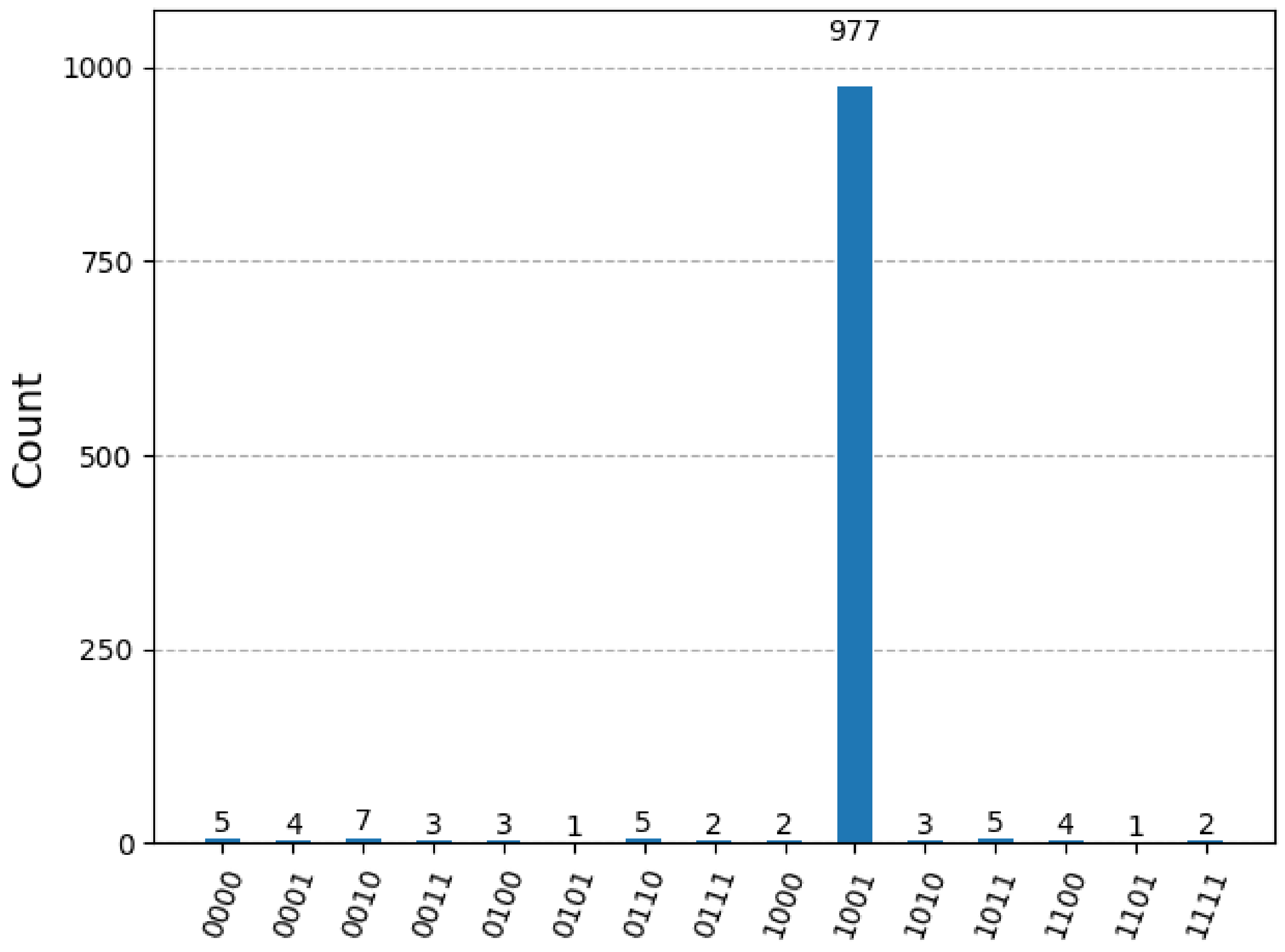


Figure 1: Simulation result using StatevectorSampler

Also, we transpile the quantum circuit and simulate its execution using the fake backend **FakeSherbrooke**, but with **SamplerV2** this time:

```
#Simulation using SamplerV2
backend = FakeSherbrooke()
shots=1024
pm = generate_preset_pass_manager(backend=backend, optimization_level=3)
transpiled_qc = pm.run(qc)
sampler = SamplerV2(mode=backend)
job = sampler.run([transpiled_qc],shots=shots)
pub_result = job.result()[0]
counts = pub_result.data.c.get_counts()
plot_histogram(counts)
```

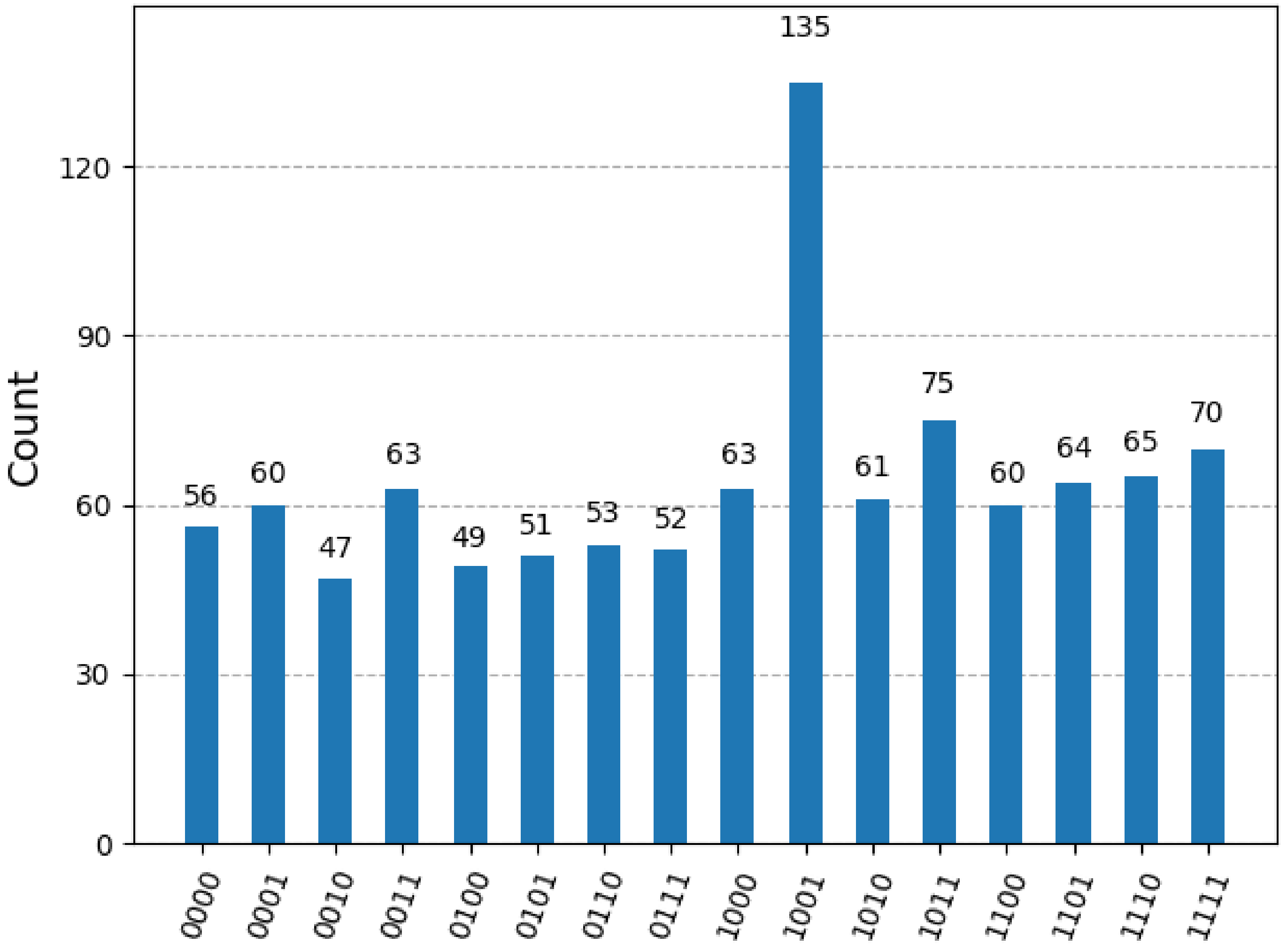


Figure 2: Simulation result using SamplerV2

We observe that the results are a little more noisy, because the noise model from the fake backend has been incorporated.

Finally, we execute the Grover's algorithm in a real IBM Quantum Computer:

```
service = QiskitRuntimeService(
    channel='ibm_quantum',
    token='QISKIT_TOKEN>'
)

shots=1024

backend = service.least_busy(operational=True, simulator=False)
pm = generate_preset_pass_manager(backend=backend, optimization_level=1)
transpiled_qc = pm.run(qc)
sampler = SamplerV2(backend)
job = sampler.run([transpiled_qc])
pub_result = job.result()[0]
counts = pub_result.data.c.get_counts()
plot_histogram(counts)
```

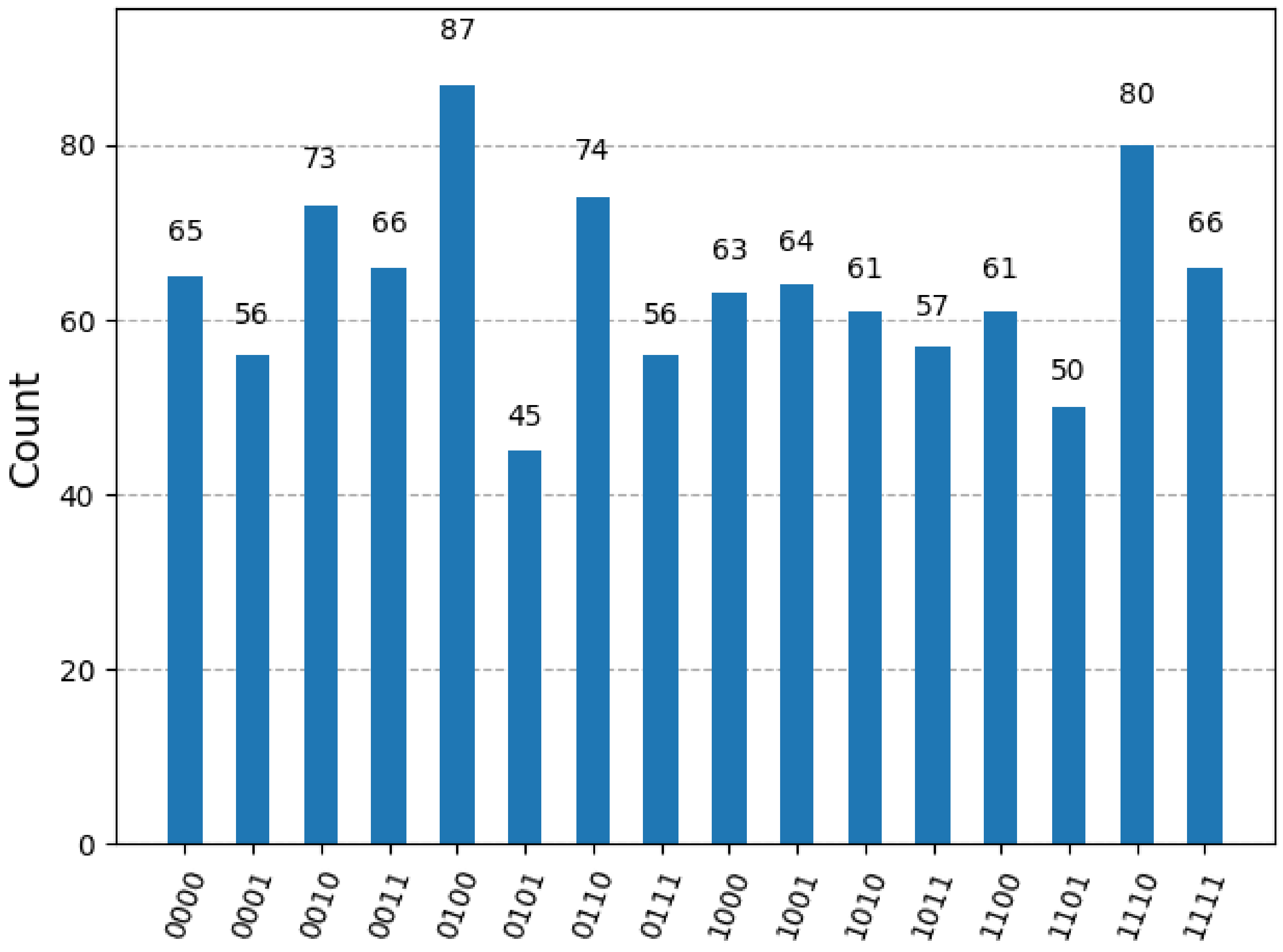


Figure 3: Result after execution on a real quantum computer

We observe that when running Grover's algorithm for 4 qubits on a real IBM Quantum computer, the results are unclear and not consistent.

You will find all of the code used for this exercise in this repository: https://github.com/topalidisd-qcqt-duth/qy3-assignment-2
You can also find all of the code below:

```
from qiskit import *
import numpy as np
from qiskit_ibm_runtime.fake_provider import FakeSherbrooke
from qiskit.visualization import plot_histogram
from qiskit_ibm_runtime import QiskitRuntimeService, SamplerV2
from qiskit.primitives import StatevectorSampler
from qiskit.transpiler import generate_preset_pass_manager


def oracle_operator(s):

    n = len(s)

    qc = QuantumCircuit(n+1)

    for i, bit in enumerate(s):
        if bit == '0':
            qc.x(i)

    qc.mcx(list(range(n)), n)

    for i, bit in enumerate(s):
        if bit == '0':
            qc.x(i)

    return qc


def diffusion_operator(n):

    qc = QuantumCircuit(n)

    qc.h(list(range(n)))
    qc.x(list(range(n)))
    qc.h(n-1)
    qc.mcx(list(range(n-1)), n-1)
    qc.h(n-1)
    qc.x(list(range(n)))
    qc.h(list(range(n)))

    return qc


def grovers_algorithm(oracle, n):

    #Step 1
    qc = QuantumCircuit(n+1, n)

    #Step 2
    qc.x(n)
    qc.h(n)


    #Step 3
    qc.h(list(range(n)))

    #Step 4
    N = int(np.ceil(np.pi / 4 * np.sqrt(2**n))) -1

    #Step 5
    for _ in range(N):
        qc.barrier()
        qc.compose(oracle, range(n+1), inplace=True)
        qc.barrier()
        qc.compose(diffusion_operator(n), range(n), inplace=True)

    #Step 6
    qc.measure(list(range(n)) , reversed(list(range(n))))

    return qc
```

```
#Example required by the assignment
oracle = oracle_operator("1001")
qc = grovers_algorithm(oracle, 4)
qc.draw('mpl', fold=False)

#Simulation using StatevectorSampler
pm = generate_preset_pass_manager(optimization_level=3)
transpiled_qc = pm.run(qc)
statevectorSampler = StatevectorSampler()
job = statevectorSampler.run([transpiled_qc])
pub_result = job.result()[0]
counts = pub_result.data.c.get_counts()
plot_histogram(counts)

#Simulation using SamplerV2
backend = FakeSherbrooke()
shots=1024
pm = generate_preset_pass_manager(backend=backend, optimization_level=3)
transpiled_qc = pm.run(qc)
sampler = SamplerV2(mode=backend)
job = sampler.run([transpiled_qc],shots=shots)
pub_result = job.result()[0]
counts = pub_result.data.c.get_counts()
plot_histogram(counts)

#Execution on a real quantum computer
service = QiskitRuntimeService(
    channel='ibm_quantum',
    token='QISKIT_TOKEN>'
)

shots=1024

backend = service.least_busy(operational=True, simulator=False)
pm = generate_preset_pass_manager(backend=backend, optimization_level=1)
transpiled_qc = pm.run(qc)
sampler = SamplerV2(backend)
job = sampler.run([transpiled_qc], shots=shots)
pub_result = job.result()[0]
counts = pub_result.data.c.get_counts()
plot_histogram(counts)
```