data

June 14, 2024

In this first block of code, we will simulate Grover's Algorithm for 4 qubits with 1 solution '1110'. To implement Grover's algorithm we will need a circuit with n qubits representing the input space, and 1 auxiliary qubit to mark the solutions.

[6]:
```
from qiskit import QuantumCircuit, transpile
from qiskit_aer import Aer
import numpy as np
from qiskit.visualization import plot_histogram
from qiskit.circuit.library import GroverOperator
import matplotlib.pyplot as plt
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

def oracle():
    qc = QuantumCircuit(5)
    """
    #qc.x(0)
    qc.ccx(0, 1, 4)
    qc.ccx(2, 4, 5)
    qc.ccx(3, 5, 6)
    qc.ccx(2, 4, 5)
    qc.ccx(0, 1, 4)
    #qc.x(0)
    """
    qc.x(0)
    mcx = MCXGate(num_ctrl_qubits=4)
    qc.append(mcx, list(range(5)))
    qc.x(0)


    return qc

# Define the Grover's diffusion operator
def diffusion_operator():
    qc = QuantumCircuit(5)
    qc.h(range(4))
    qc.x(range(4))
    mcx = MCXGate(num_ctrl_qubits=4)
```

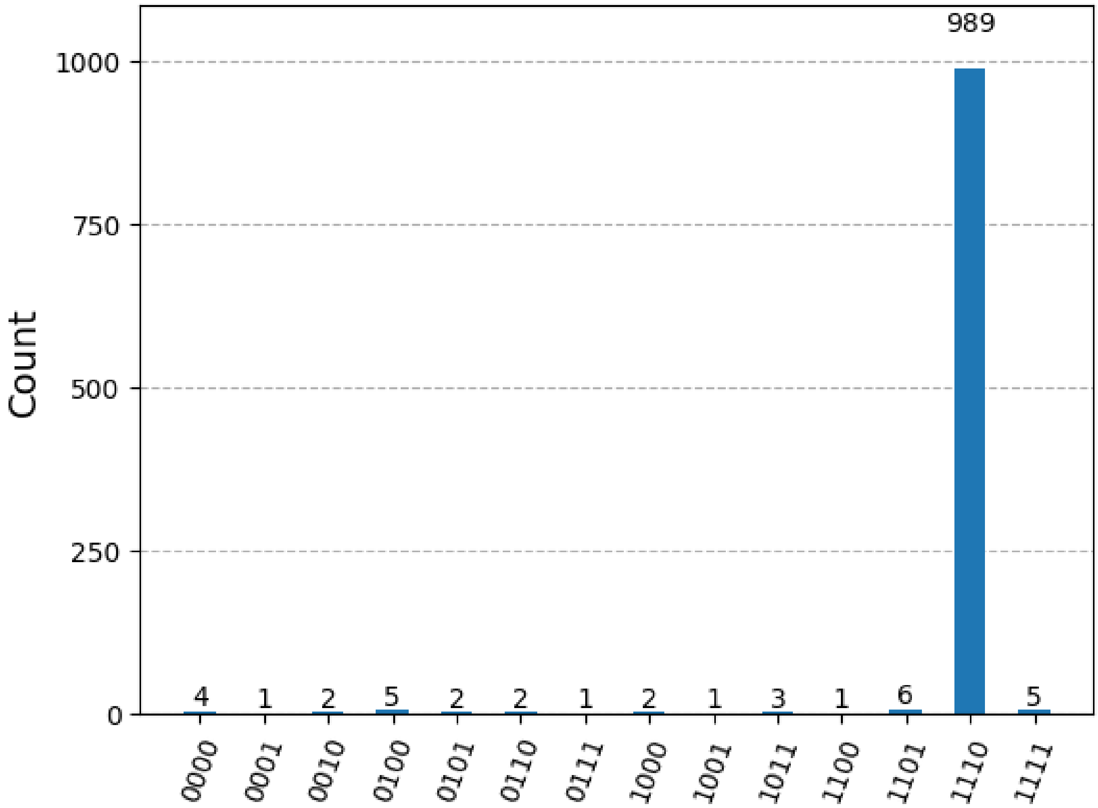
```python
    qc.append(mcx, list(range(5)))
    qc.x(range(4))
    qc.h(range(4))
    return qc

# Create the Grover's algorithm circuit
grover_circuit = QuantumCircuit(5, 4)

# Initialize the qubits to a uniform superposition state
grover_circuit.h(range(4))

grover_circuit.x(4)
grover_circuit.h(4)
# Append the oracle and diffusion operator
r = int(np.floor(np.pi/4*np.sqrt(2**(4))))
for i in range(r):
    oracle_gate = oracle().to_gate()
    grover_circuit.append(oracle_gate, range(5))

    diffusion_gate = diffusion_operator().to_gate()
    grover_circuit.append(diffusion_gate, range(5))

# Measure the qubits
grover_circuit.measure(range(4), range(4))
grover_circuit.draw()
# Simulate the circuit
backend = Aer.get_backend('qasm_simulator')
#transpiled_circuit = transpile(grover_circuit, backend)

nc = transpile(grover_circuit,backend)
job = backend.run(nc)
result = job.result()


# Get the counts of the measurement results
counts = result.get_counts(grover_circuit)

# Print and plot the results
print(counts)
plot_histogram(counts)
plt.show()
```

[6]: <qiskit.circuit.instructionset.InstructionSet at 0x1247defa940>

[6]: <qiskit.circuit.instructionset.InstructionSet at 0x1247defae50>

[6]: <qiskit.circuit.instructionset.InstructionSet at 0x1247d28eca0>

```
[6]: <qiskit.circuit.instructionset.InstructionSet at 0x1247f5ac8b0>

[6]: <qiskit.circuit.instructionset.InstructionSet at 0x1247df050a0>

[6]: <qiskit.circuit.instructionset.InstructionSet at 0x1247df054c0>

[6]: <qiskit.circuit.instructionset.InstructionSet at 0x1247df524f0>

[6]: <qiskit.circuit.instructionset.InstructionSet at 0x1247df52fa0>

[6]: <qiskit.circuit.instructionset.InstructionSet at 0x1243ca6df40>

[6]: <qiskit.circuit.instructionset.InstructionSet at 0x1243daf3f10>

[6]:                                               »
q_0:   H     0              0              0              0             »
                                                                         »
q_1:   H     1              1              1              1             »
                                                                         »
q_2:   H     2 circuit-189  2 circuit-192  2 circuit-195  2 circuit-198  »
                                                                         »
q_3:   H     3              3              3              3             »
                                                                       »
q_4:   X   H  4              4              4              4             »
                                                »
c: 4/                                           »
                                                                              »
«
«q_0:  0              0              M
«
«q_1:  1              1                M
«
«q_2:  2 circuit-201  2 circuit-204      M
«
«q_3:  3              3                    M
«
«q_4:  4              4
«
«c: 4/
«                                       0  1  2  3

{'0101': 2, '1110': 989, '0010': 2, '1011': 3, '1101': 6, '0100': 5, '1100': 1,
'1111': 5, '0000': 4, '0111': 1, '1000': 2, '0110': 2, '1001': 1, '0001': 1}
[6]:
```



Now we will write a function for Grover's algortihm with as input the number of input qubits, and an array of arrays representing the solutions.

[40]:
```
from qiskit import QuantumCircuit, transpile
from qiskit_aer import Aer
import numpy as np
from qiskit.visualization import plot_histogram
from qiskit.circuit.library import GroverOperator
import matplotlib.pyplot as plt
from IPython.core.interactiveshell import InteractiveShell
from qiskit.visualization import plot_bloch_multivector

InteractiveShell.ast_node_interactivity = "all"

def grover(n_bits, solutions):

    oracle, osv = oracle_operator(n_bits, solutions)
    diffusor, dsv = diffusion_operator(n_bits)

    grover_circuit = QuantumCircuit(n_bits+1, n_bits)
```

```python
    # Initialize the qubits to a uniform superposition state
    grover_circuit.h(range(n_bits))

    grover_circuit.x(n_bits)
    grover_circuit.h(n_bits)
    # Append the oracle and diffusion operator
    r = int(np.floor(np.pi/4*np.sqrt(2**n_bits/len(solutions))))

    for i in range(r):
        oracle_gate = oracle.to_gate()
        grover_circuit.append(oracle_gate, range(n_bits+1))

        diffusion_gate = diffusor.to_gate()
        grover_circuit.append(diffusion_gate, range(n_bits+1))

    # Measure the qubits
    grover_circuit.measure(range(n_bits), range(n_bits))
    grover_circuit.draw()
    # Simulate the circuit
    backend = Aer.get_backend('qasm_simulator')
    #transpiled_circuit = transpile(grover_circuit, backend)

    nc = transpile(grover_circuit,backend)
    job = backend.run(nc)
    result = job.result()


    # Get the counts of the measurement results
    counts = result.get_counts(grover_circuit)

    # Print and plot the results
    print(counts)
    plot_histogram(counts)
    plt.show()
    return counts, osv, dsv

def oracle_operator(n_bits, solutions):
    qc = QuantumCircuit(n_bits+1)

    for solution in solutions:
        save = []
        for i in range(n_bits):
            if not solution[-1-i]:
                qc.x(i)
                save.append(i)

        mcx = MCXGate(num_ctrl_qubits=n_bits)
```

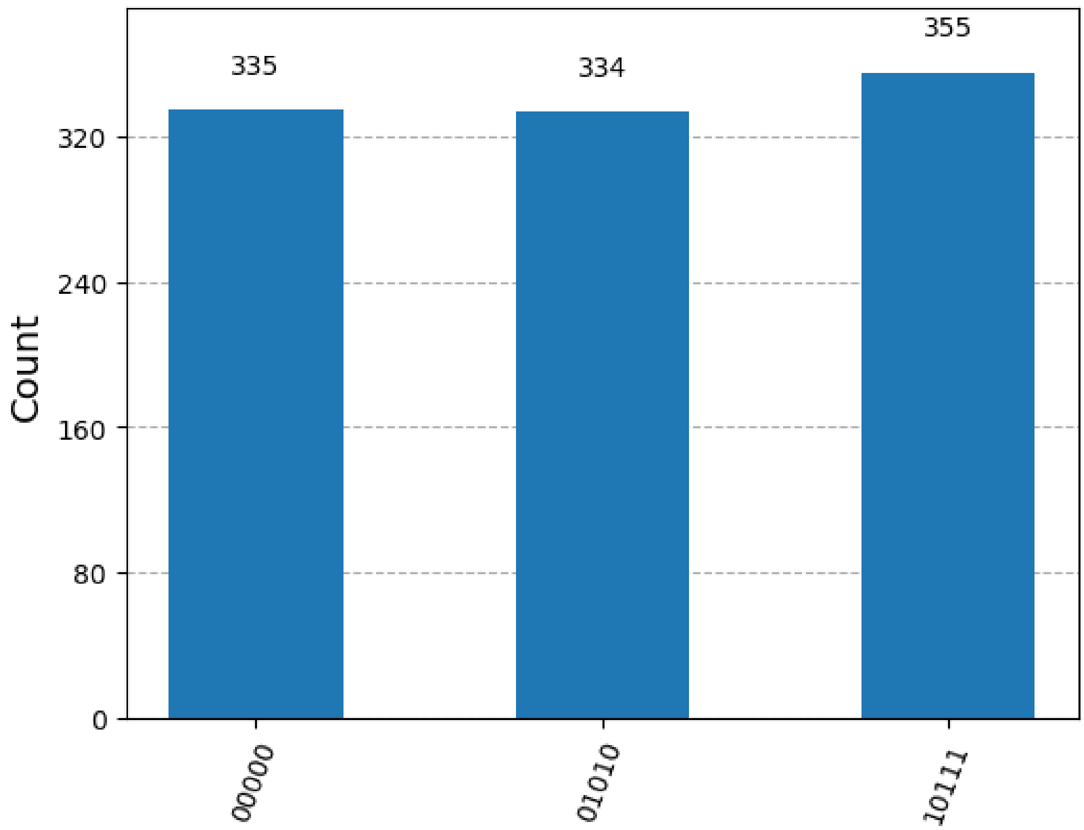
```
        qc.append(mcx, list(range(n_bits+1)))

        for i in save:
            qc.x(i)

    backend = Aer.get_backend("statevector_simulator")
    nc = transpile(qc,backend)
    job = backend.run(nc)
    result = job.result()
    statevec = result.get_statevector()

    return qc, statevec

# Define the Grover's diffusion operator
def diffusion_operator(n_bits):
    qc = QuantumCircuit(n_bits+1)
    qc.h(range(n_bits))
    qc.x(range(n_bits))
    mcx = MCXGate(num_ctrl_qubits=n_bits)
    qc.append(mcx, list(range(n_bits+1)))
    qc.x(range(n_bits))
    qc.h(range(n_bits))

    backend = Aer.get_backend("statevector_simulator")
    nc = transpile(qc,backend)
    job = backend.run(nc)
    result = job.result()
    statevec = result.get_statevector()

    return qc, statevec
```

In the next code block we will run the algorithm.

[42]:
```
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
counts, osv, dsv = grover(5, [[1,0,1,1,1],[0,0,0,0,0],[0,1,0,1,0]])
plot_histogram(counts)
plt.show()
```

```
{'00000': 335, '01010': 334, '10111': 355}
```

[42]:



NOW WITH ORACLE SET_UP TO SEE STATEVECS

```
[ ]:
```