SCS 43XX

Quantum Mechanics in Computing

University of Colombo, School of Computing

Labsheet 04

Circuits in Qiskit

1 Instructions

1.1 Multi-Qubit States and Entanglement

A single qubit can be in a superposition of $|0\rangle$ and $|1\rangle$, but with multiple qubits, we can create **entanglement**—a quantum correlation where measuring one qubit affects the other.

A common entangled state is the **Bell state**:

$$|\Phi^+\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}}$$

This state can be created using a Hadamard and a CNOT gate.

```
from qiskit import QuantumCircuit

qc = QuantumCircuit(2)
qc.h(0)       # Hadamard on first qubit
qc.cx(0, 1)   # CNOT: first qubit controls second
qc.measure_all()
qc.draw()
```

1.2 Multi-Qubit Gates

Quantum circuits often require multi-qubit gates:

- **CNOT (CX)**: Flips the target qubit if the control qubit is $|1\rangle$.
- **Toffoli (CCX)**: A 3-qubit gate that flips the target qubit only if both control qubits are $|1\rangle$.
- **SWAP**: Swaps the states of two qubits.

```
qc = QuantumCircuit(3)
qc.ccx(0, 1, 2)  # Toffoli: Qubit 2 flips if Qubit 0 and 1 are |1>
qc.swap(1, 2)    # Swap states of Qubit 1 and Qubit 2
qc.draw()
```

1.3 Quantum Measurement and Visualization

Measurements collapse quantum states into classical bits. Results can be analyzed using histograms.

```
from qiskit.visualization import plot_histogram

qc.measure_all()
backend = Aer.get_backend('qasm_simulator')
job = execute(qc, backend, shots=1024)
result = job.result()
plot_histogram(result.get_counts())
```

1.4 Quantum Fourier Transform (QFT)

The **Quantum Fourier Transform (QFT)** is a quantum version of the discrete Fourier transform (DFT). It is used to analyze periodicity in quantum states and plays an essential role in algorithms like **Shor's factoring algorithm**.

The QFT on n qubits is defined as:

$$QFT_n = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} \sum_{k=0}^{n-1} \omega^{jk} |j\rangle\langle k|$$

where $\omega = e^{2\pi i/n}$.

For an efficient implementation, we perform a series of **Hadamard** gates and **controlled phase** gates: - The **Hadamard** gate creates superposition. - The **controlled phase shift** gates are used to entangle the qubits in a superposition of the frequency states.

Here is the correct code for a 3-qubit QFT circuit:

```
from qiskit import QuantumCircuit
import numpy as np

def qft(n):
    qc = QuantumCircuit(n)
    for i in range(n):
        qc.h(i)
        for j in range(i+1, n):
            qc.cp(np.pi/2**(j-i), j, i)  # Controlled phase shift
    qc.barrier()
    for i in range(n//2):
        qc.swap(i, n-i-1)
    return qc

qc = qft(3)
qc.draw()
```

Key Points: 1. **Hadamard gates** (H) are used to create superposition of all states. 2. **Controlled Phase gates** (cp) introduce relative phases between the qubits based on their positions. 3. The final **swap** ensures that the QFT algorithm works in the reverse order of the qubits.

1.5 Deutsch-Jozsa Algorithm

The **Deutsch-Jozsa algorithm** solves the problem of determining whether a given function is **constant** or **balanced** with just **one query** to the oracle, using quantum parallelism. This is exponentially faster than the best classical algorithm, which would require multiple evaluations.

The algorithm uses the following steps: 1. Prepare n qubits in a superposition state. 2. Apply the **oracle** (a unitary operation that implements the function). 3. Apply a series of **Hadamard gates** to the qubits. 4. Measure the qubits to determine whether the function is constant or balanced.

Here's an example for implementing the Deutsch-Jozsa algorithm using a balanced oracle:

```
qc = QuantumCircuit(3)
qc.h([0, 1])  # Superposition
qc.x(2)       # Oracle qubit
qc.h(2)
qc.barrier()

# Example of a balanced oracle
qc.cx(0, 2)
qc.cx(1, 2)

qc.barrier()
qc.h([0, 1])
qc.measure_all()
qc.draw()
```

Key Points: - The algorithm makes use of quantum **parallelism** to evaluate all inputs simultaneously. - A balanced function will cause destructive interference, leaving a clear result after the Hadamard gates are applied to the qubits. - A constant function will not result in interference, so the outcome will be the same.

2 Tasks

2.1 Activity 1: Creating a Bell State and Verifying Entanglement

Objective: Construct a Bell state and verify entanglement using measurements.

Steps:

1. Create a 2-qubit circuit.
2. Apply a Hadamard gate to Qubit 0.
3. Apply a CNOT gate (Qubit 0 controls Qubit 1).
4. Measure both qubits and execute on a simulator.
5. Plot the results.

2.2 Activity 2: Creating a GHZ State

Objective: Construct a 3-qubit **GHZ** state:

$$|GHZ\rangle = \frac{|000\rangle + |111\rangle}{\sqrt{2}}$$

Steps:

1. Create a 3-qubit circuit.
2. Apply a Hadamard gate to Qubit 0.
3. Apply a CNOT from Qubit 0 to Qubit 1.
4. Apply a CNOT from Qubit 1 to Qubit 2.
5. Measure all qubits and analyze the results.

2.3 Activity 3: Quantum Teleportation

Objective: Implement the quantum teleportation protocol to transfer a qubit state using entanglement.

Steps:

1. Set up 3 qubits:
   - Qubit 0: State to be teleported.
   - Qubit 1: Part of an entangled pair with Qubit 2.
   - Qubit 2: Receiver.
2. Create a Bell pair between Qubits 1 and 2.
3. Perform a Bell measurement on Qubits 0 and 1.
4. Use classical communication and correction gates (X, Z) to recover the state.

2.4 Activity 4: Implementing the Quantum Fourier Transform

Objective: Implement and test the Quantum Fourier Transform on a 3-qubit system.

Steps:

1. Construct a function to apply QFT to n qubits.
2. Apply the QFT circuit to a superposition state.
3. Measure the circuit and analyze the results.

2.5 Activity 5: Implementing the Deutsch-Jozsa Algorithm

Objective: Implement the Deutsch-Jozsa algorithm for a function that is either **constant** or **balanced**.

Steps:

1. Construct a quantum circuit with three qubits.
2. Apply Hadamard gates to the first two qubits.
3. Implement an oracle for a balanced or constant function.
4. Apply Hadamard gates again and measure the first two qubits.
5. Interpret the results.

2.6 Activity 5: Implementing the Deutsch-Jozsa Algorithm

Objective: Implement the Deutsch-Jozsa algorithm for a function that is either **constant** or **balanced**.

Steps:

1. Construct a quantum circuit with three qubits.
2. Apply Hadamard gates to the first two qubits.
3. Implement an oracle for a balanced or constant function.
4. Apply Hadamard gates again and measure the first two qubits.
5. Interpret the results.

2.7 Running on a Real Quantum Computer

If you have access to IBM Quantum, you can execute circuits on a real device.

```
from qiskit.providers.ibmq import IBMQ

IBMQ.save_account('API_TOKEN')  # Save your IBM Quantum token
provider = IBMQ.load_account()
backend = provider.get_backend('ibmq_qasm_simulator')

job = execute(qc, backend, shots=1024)
result = job.result()
plot_histogram(result.get_counts())
```