**Lab Manual: Basic Quantum Algorithms in Qiskit**

**Step 1: Theoretical Background**

Quantum algorithms leverage quantum mechanics principles to solve problems more efficiently than classical algorithms. Key concepts include:

* **Superposition**: A quantum system can exist in multiple states at once.
* **Entanglement**: Qubits can be correlated such that the state of one qubit depends on the state of another.
* **Quantum Gates**: Operations that manipulate qubits to perform calculations.

**Step 2: Quantum Algorithms**

**2.1: Deutsch-Josza Algorithm**

**Description**: Determines whether a given function is constant or balanced.

**Implementation**:

from qiskit import QuantumCircuit

from qiskit\_aer import AerSimulator

import matplotlib.pyplot as plt

def deutsch\_jozsa\_oracle(circuit, n, constant=True):

if constant:

pass # Constant function

else:

for i in range(n):

circuit.cx(i, n) # Balanced function

def deutsch\_jozsa\_algorithm(n, constant=True):

circuit = QuantumCircuit(n + 1, n)

circuit.x(n) # Initialize output to |1⟩

circuit.h(range(n + 1)) # Apply Hadamard gates

deutsch\_jozsa\_oracle(circuit, n, constant)

circuit.h(range(n)) # Apply Hadamard gates again

circuit.measure(range(n), range(n))

return circuit

# Set the number of qubits

n = 3

circuit = deutsch\_jozsa\_algorithm(n, constant=False)

# Draw and execute the circuit

print("Deutsch-Josza Circuit:")

print(circuit.draw())

simulator = AerSimulator()

result = simulator.run(circuit).result()

counts = result.get\_counts()

# Plotting results

plt.bar(counts.keys(), counts.values(), color='skyblue')

plt.title('Deutsch-Josza Algorithm Results')

plt.xlabel('States')

plt.ylabel('Counts')

plt.show()

**2.2: Grover's Algorithm**

**Description**: Searches for a marked item in an unsorted database with quadratic speedup.

**Implementation**:

def grover\_oracle(circuit, n, marked\_state):

"""Oracle that marks the desired state."""

# Assuming marked\_state is an integer representing binary number

binary\_state = format(marked\_state, f'0{n}b')

for i, bit in enumerate(binary\_state):

if bit == '0':

circuit.x(i) # Flip to |1⟩ if 0

circuit.h(n - 1) # Apply Hadamard to the last qubit

circuit.mct(list(range(n)), n - 1) # Multi-controlled Toffoli

circuit.h(n - 1) # Apply Hadamard to the last qubit

for i, bit in enumerate(binary\_state):

if bit == '0':

circuit.x(i) # Flip back to |0⟩

def grover\_algorithm(n, marked\_state):

circuit = QuantumCircuit(n)

# Apply Hadamard gates to all qubits

circuit.h(range(n))

# Apply the oracle for Grover's algorithm

grover\_oracle(circuit, n, marked\_state)

# Apply diffusion operator

circuit.h(range(n))

circuit.x(range(n))

circuit.h(n - 1)

circuit.mct(list(range(n - 1)), n - 1) # Multi-controlled Toffoli

circuit.h(n - 1)

circuit.x(range(n))

circuit.h(range(n))

circuit.measure\_all()

return circuit

# Set the number of qubits and the marked state

n = 3 # Number of qubits

marked\_state = 5 # Example: |101⟩ is marked

circuit = grover\_algorithm(n, marked\_state)

# Draw and execute the circuit

print("Grover's Circuit:")

print(circuit.draw())

result = simulator.run(circuit).result()

counts = result.get\_counts()

# Plotting results

plt.bar(counts.keys(), counts.values(), color='skyblue')

plt.title('Grover\'s Algorithm Results')

plt.xlabel('States')

plt.ylabel('Counts')

plt.show()

**2.3: Quantum Fourier Transform (QFT)**

**Description**: The quantum analogue of the classical Fourier transform.

**Implementation**:

def qft(circuit, n):

"""Applies the Quantum Fourier Transform to n qubits."""

for j in range(n):

circuit.h(j)

for k in range(j + 1, n):

circuit.cp(np.pi / 2\*\*(k - j), k, j) # Controlled phase rotation

def quantum\_fourier\_transform(n):

circuit = QuantumCircuit(n)

qft(circuit, n)

circuit.measure\_all() # Measure all qubits

return circuit

# Set the number of qubits

n = 3

circuit = quantum\_fourier\_transform(n)

# Draw and execute the circuit

print("QFT Circuit:")

print(circuit.draw())

result = simulator.run(circuit).result()

counts = result.get\_counts()

# Plotting results

plt.bar(counts.keys(), counts.values(), color='skyblue')

plt.title('Quantum Fourier Transform Results')

plt.xlabel('States')

plt.ylabel('Counts')

plt.show()

**2.4: Shor's Algorithm**

**Description**: Shor's algorithm efficiently factors large integers, which is a significant problem in cryptography.

**Circuit Diagram**: The implementation involves creating a quantum circuit to perform the modular exponentiation and then applying the QFT.

**Implementation**:

python

RunCopy

from qiskit import Aer, transpile

from qiskit.algorithms import Shor

def shors\_algorithm(N):

"""Implements Shor's algorithm for integer factorization."""

shor = Shor()

result = shor.factor(N)

return result

# Example: Factor the number 15

N = 15

factors = shors\_algorithm(N)

print(f"Factors of {N}: {factors}")

**2.5: Variational Quantum Eigensolver (VQE)**

**Description**: VQE is used to find the ground state energy of a quantum system using hybrid quantum-classical algorithms.

**Circuit Diagram**: The VQE typically uses parameterized quantum circuits and classical optimization techniques.

**Implementation**:

python

RunCopy

from qiskit.circuit import QuantumCircuit

from qiskit.primitives import Sampler

from qiskit.algorithms import VQE

from qiskit.algorithms.optimizers import SLSQP

from qiskit.quantum\_info import Pauli

from qiskit.opflow import PauliExpectation, CircuitSampler, StateFn

def vqe\_example():

"""Implements a basic VQE to find the ground state energy of H2."""

# Define the Hamiltonian for H2

hamiltonian = PauliExpectation().convert(StateFn(Pauli("ZZ")))

# Create a parameterized circuit

ansatz = QuantumCircuit(2)

ansatz.ry(0.1, 0)

ansatz.ry(0.1, 1)

# Set up the optimizer

optimizer = SLSQP(maxiter=100)

# Set up VQE

vqe = VQE(ansatz, optimizer=optimizer)

# Execute VQE

result = vqe.compute\_minimum\_eigenvalue(hamiltonian)

return result

energy = vqe\_example()

print(f"Estimated ground state energy: {energy.eigenvalue.real:.4f}")

**Step 3: Applications**

1. **Deutsch-Josza Algorithm**:
   * Cryptography and optimization problems.
2. **Grover's Algorithm**:
   * Search problems and optimization tasks.
3. **Quantum Fourier Transform (QFT)**:
   * Signal processing and Shor's algorithm.
4. **Shor's Algorithm**:
   * Cryptography, particularly in breaking RSA encryption.
5. **Variational Quantum Eigensolver (VQE)**:
   * Quantum chemistry and materials science for finding molecular ground states.

**Step 4: Analyze the Results**

* **Deutsch-Josza Algorithm**:
  + Constant function returns all zeros; balanced function shows a distribution.
* **Grover's Algorithm**:
  + The marked state should have the highest count.
* **Quantum Fourier Transform**:
  + The measurement will reflect the transformed states.
* **Shor's Algorithm**:
  + The result will show the factors of the input integer.
* **Variational Quantum Eigensolver (VQE)**:
  + The estimated ground state energy of the system.

**Conclusion**

This lab manual demonstrates how to implement several fundamental quantum algorithms using Qiskit. By leveraging quantum principles, these algorithms provide significant speedup and efficiency in solving various problems compared to classical algorithms.

**Further Exploration**

1. **Modify Oracles**: Experiment with different oracles in Grover's algorithm.
2. **Explore Other Algorithms**: Implement additional algorithms like Quantum Phase Estimation.
3. **Analyze Different Input Sizes**: Test the algorithms with varying numbers of qubits to observe performance differences.