

MDS572B: QUANTUM MACHINE LEARNING

OBSERVATION NOTEBOOK

Name: Neelanjan Dutta

Roll No: 2448040

Class: 5MDS

LAB No. 4: Implement the DJ algorithm using IBM Simulator and/or Qiskit

Question:

Implement a DJ algorithm using Qiskit

Objective:

The objective of this lab is to design and implement the Deutsch-Jozsa (DJ) quantum algorithm. This involves creating generalized functions for two types of oracles—**constant** and **balanced**—and building the main DJ circuit that uses them. The goal is to visually demonstrate how the algorithm's circuit is constructed for both types of oracles by prompting the user to enter the number of qubits (n) and displaying the resulting circuit diagrams.

Draft Plan:

Program description:

This program uses Qiskit to build the Deutsch-Jozsa algorithm. It defines Python functions to create a constant oracle (which applies an X gate or no gate to the ancilla) and a balanced oracle (which implements a parity function using CNOT gates). The main program prompts the user for the number of input qubits (n). It then constructs the full DJ circuit for three cases: a constant function $f(x) = 0$, a constant function $f(x) = 1$, and a balanced parity function. Finally, it uses **matplotlib.pyplot** to draw and display all three resulting circuit diagrams in a single window.

Program logic:

1. **Initialization:** Import **QuantumCircuit** and **transpile** from **qiskit**, and **matplotlib.pyplot** for plotting.
2. **Constant Oracle function:**
 - ❖ Define **constant_oracle(n, output_bit)**
 - ❖ It creates a QuantumCircuit of $n+1$ qubits .

- ❖ If **output_bit** is 1, it applies a single X gate to the ancilla qubit (at index n) to make the function output 1.

3. Balanced oracle function:

- ❖ Define **balanced_oracle_parity(n)**.
- ❖ It creates a circuit of n+1 qubits
- ❖ It iterates from i=0 to n-1 and applies a CX (CNOT) gate from each input qubit i to the ancilla qubit n. This implements the parity function, which is a balanced function.

4. Deutsch – Jozsa algorithm :

- ❖ Define **deutsch_jozsa(n, oracle_circ)** .
- ❖ Create a circuit with n+1 qubits and n classical bits.
- ❖ **Initialization:** Apply an X gate to the ancilla (n) to set it to $|1\rangle$, then an H gate to put it in the $|-\rangle$ state. Apply H gates to all n input qubits to create a uniform superposition.
- ❖ **Oracle:** Append the provided **oracle_circ** to the circuit.
- ❖ **Decomposition:** Use **qc.decompose([oracle_circ.name])** to expand the oracle box and show its internal gates (like X or CX) in the final diagram.
- ❖ **Final Hadamards:** Apply H gates again to the n input qubits.
- ❖ **Measurement:** Measure each of the n input qubits into the n classical bits.

5. Main Execution:

- ❖ The **if __name__ == "__main__":** block prompts the user to enter n.
- ❖ It generates all three DJ circuits by calling the functions above (Constant-0, Constant-1, Balanced).
- ❖ It draws each circuit to its designated panel (ax1, ax2, ax3) using the **draw('mpl')** function and sets a title for each.
- ❖ It displays the final plot window with **plt.show()**.

Program:

P.T.O.

```
1 from qiskit import QuantumCircuit, transpile
2 import matplotlib.pyplot as plt
```

```
1 def constant_oracle(n, output_bit=0):
2     qc = QuantumCircuit(n + 1, name=f"const_{output_bit}")
3     if output_bit == 1:
4         qc.x(n) # Flip the output qubit to 1
5     return qc
```

```
1 def balanced_oracle_parity(n):
2     qc = QuantumCircuit(n + 1, name="balanced_parity")
3     for i in range(n):
4         qc.cx(i, n) # CNOT from input i to output n
5     return qc
```

```
1 def deutsch_jozsa(n, oracle_circ):
2     qc = QuantumCircuit(n + 1, n)
3     # Initialization
4     qc.x(n)
5     qc.h(n)
6     for i in range(n):
7         qc.h(i)
8     # Oracle
9     qc.append(oracle_circ.to_instruction(), range(n + 1))
10    # Decompose the oracle to show CNOTs, etc.
11    qc = qc.decompose([oracle_circ.name])
12    # Final Hadamards
13    for i in range(n):
14        qc.h(i)
15    # Measure
16    for i in range(n):
17        qc.measure(i, i)
18    return qc
```

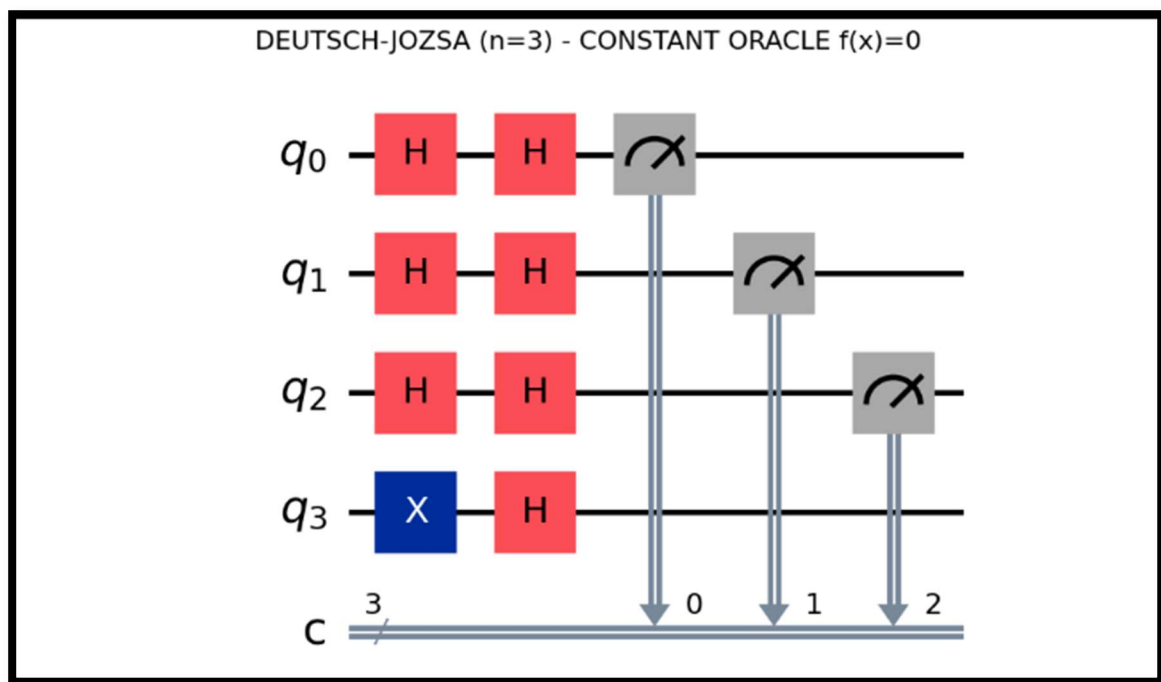
```

1 if __name__ == "__main__":
2     try:
3         # Get user input for the number of qubits
4         n_str = input("Enter the number of input qubits (n) ")
5         n = int(n_str)
6
7         if n < 1:
8             print("Error: Number of qubits (n) must be 1 or greater.")
9         else:
10            print(f"\nGenerating circuits for n = {n}\n")
11
12            # Create all 3 circuits
13            oracle_const_0 = constant_oracle(n, output_bit=0)
14            dj_circ_const_0 = deutsch_jozsa(n, oracle_const_0)
15
16            oracle_const_1 = constant_oracle(n, output_bit=1)
17            dj_circ_const_1 = deutsch_jozsa(n, oracle_const_1)
18
19            oracle_balanced = balanced_oracle_parity(n)
20            dj_circ_balanced = deutsch_jozsa(n, oracle_balanced)
21
22            # Creating a Matplotlib figure with 3 subplots
23            fig, (ax1, ax2, ax3) = plt.subplots(3, 1, figsize=(10, 12 + n*2))
24
25            # Circuit 1: Constant f(x)=0
26            dj_circ_const_0.draw(output='mpl', ax=ax1)
27            ax1.set_title(f"DEUTSCH-JOZSA (n={n}) - CONSTANT ORACLE f(x)=0", fontsize=16)
28
29            # Circuit 2: Constant f(x)=1
30            dj_circ_const_1.draw(output='mpl', ax=ax2)
31            ax2.set_title(f"DEUTSCH-JOZSA (n={n}) - CONSTANT ORACLE f(x)=1", fontsize=16)
32
33            # Circuit 3: Balanced (Parity)
34            dj_circ_balanced.draw(output='mpl', ax=ax3)
35            ax3.set_title(f"DEUTSCH-JOZSA (n={n}) - BALANCED ORACLE (Parity)", fontsize=16)
36
37            # Show the plot window
38            plt.tight_layout() # Keeps plots from overlapping
39            print("Displaying circuit diagrams in a new window...")
40            plt.show()
41        except ValueError:
42            print("\nError: Invalid input. Please enter a whole number.")
43        except Exception as e:
44            print(f"\nAn error occurred: {e}")

```

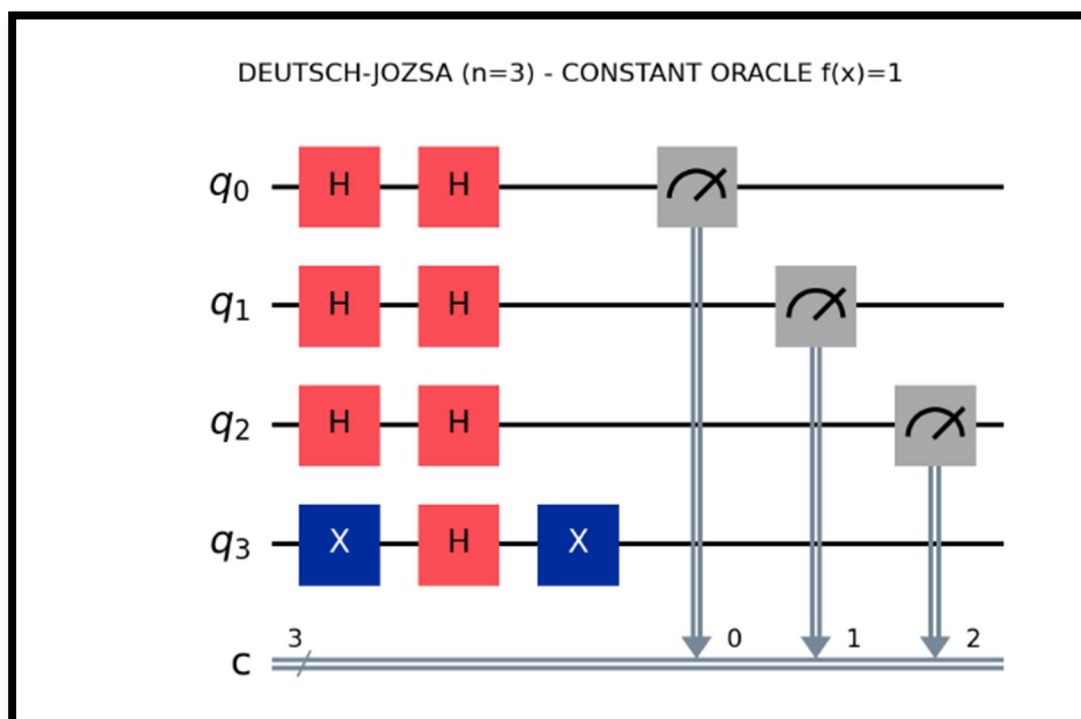
Plot 1: DEUTSCH-JOZSA (n=3) - CONSTANT ORACLE $f(x)=0$

This circuit diagram shows 4 qubits (q_0, q_1, q_2, q_3). It shows the initialization (H on inputs, X and H on ancilla q_3). The oracle part of the circuit is empty. It shows the final H gates on the inputs, followed by measurement.



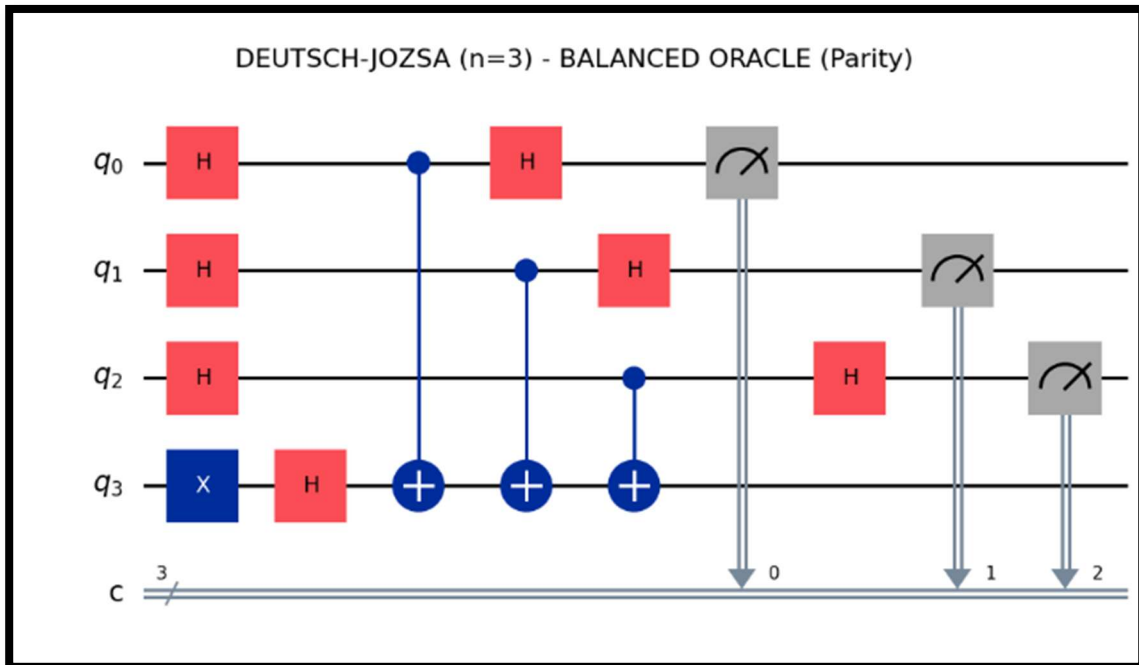
Plot 2: DEUTSCH-JOZSA (n=3) - CONSTANT ORACLE $f(x)=1$

This circuit diagram is identical to the one above, except for the oracle section. The oracle part contains a single X gate on the ancilla qubit (q_3).



Plot 3: DEUTSCH-JOZSA (n=3) - BALANCED ORACLE

This circuit diagram is identical to the others in its setup and finalization. The oracle part clearly shows three CX (CNOT) gates. The control points are on q_0 , q_1 , and q_2 , and all three target the ancilla qubit q_3 .



Conclusion:

This lab successfully implemented the Deutsch-Jozsa algorithm in Qiskit. By creating modular functions for constant and balanced oracles, the program can dynamically build the required quantum circuit based on user input for n qubits. The use of `qc.decompose()` combined with matplotlib drawing provided a clear visual distinction between the different oracle types, which are the core of the algorithm. It was visually confirmed that the $f(x)=0$ oracle is empty, the $f(x)=1$ oracle is a single X gate on the ancilla, and the balanced parity oracle is a series of CNOT gates. This exercise demonstrates how the same quantum algorithm framework can be applied to different "black box" functions to determine their global properties.
