Report on the Deutsch Algorithm Implementation in Qiskit

Mina Shiri

September 10, 2025

1 Introduction

The Deutsch Algorithm is a fundamental quantum algorithm that demonstrates the advantage of quantum computing over classical computing. It determines whether a binary function $f : \{0, 1\} \to \{0, 1\}$ is *constant* ($f(0) = f(1)$) or *balanced* ($f(0) \neq f(1)$) with a single query, compared to two queries required classically in the worst case.

2 Overview of the Deutsch Algorithm

The Deutsch Algorithm uses two qubits: an input qubit and an ancilla qubit. The steps are:

1. Apply a Hadamard gate to the input qubit to create superposition: $|0\rangle \to \frac{|0\rangle+|1\rangle}{\sqrt{2}}$.
2. Prepare the ancilla qubit in $|-\rangle = \frac{|0\rangle-|1\rangle}{\sqrt{2}}$ using an X gate followed by a Hadamard gate.
3. Apply the oracle U_f, which maps $|x\rangle|y\rangle \to |x\rangle|y \oplus f(x)\rangle$.
4. Apply another Hadamard gate to the input qubit.
5. Measure the input qubit: 0 indicates a constant function, 1 indicates a balanced function.

The provided oracle uses a CNOT gate, implementing a balanced function $f(x) = x$ ($f(0) = 0$, $f(1) = 1$).

3 Code Analysis

3.1 Importing Libraries

The code ensures Qiskit, Qiskit Aer, Matplotlib, and PyLaTeXenc are available:

```python
try:
    import qiskit
    import qiskit_aer
    import matplotlib
    import pylatexenc
except ImportError:
    print("Installing qiskit and qiskit-aer...")
    import subprocess
    import sys
    subprocess.check_call([sys.executable, "-m", "pip", "install", "qiskit", "qiskit-aer"])
finally:
    from qiskit import QuantumCircuit, transpile
    from qiskit_aer import AerSimulator
    from qiskit.visualization import plot_histogram
    import matplotlib.pyplot as plt
```

3.2 Creating the Quantum Circuit

A circuit with 2 qubits and 1 classical bit is initialized:

```
qc = QuantumCircuit(2, 1)
```

3.3 Preparing Superposition

Hadamard gate on the input qubit (q0):

```
qc.h(0)
```

X and Hadamard gates on the ancilla qubit (q1):

```
qc.x(1)
qc.h(1)
```

3.4 Oracle Implementation

The oracle is a CNOT gate, representing a balanced function:

```
oracle = QuantumCircuit(2, name='Balanced Oracle (Uf)')
oracle.cx(0, 1)
qc.append(oracle, [0, 1])
```

3.5 Final Steps and Measurement

Another Hadamard gate on the input qubit and measurement:

```
qc.h(0)
qc.measure(0, 0)
```

3.6 Simulation

The circuit is simulated with 4000 shots using AerSimulator:

```
backend = AerSimulator()
tqc = transpile(qc, backend)
job = backend.run(tqc, shots=4000)
result = job.result()
counts = result.get_counts()
```

3.7 Visualizing the Circuit

The circuit is visualized, with a fallback to text if Matplotlib fails:

```
print("\n--- Visualizing Quantum Circuit ---")
try:
    circuit_diagram = qc.draw(output='mpl', style='iqx', scale=1.1)
    circuit_diagram.show()
    print("Circuit diagram displayed in a new window.")
except Exception as e:
    print(f"Could not display matplotlib diagram automatically: {e}")
    print("Falling back to text-based diagram:")
    print(qc)
    print("-----------------------------------\n")
```

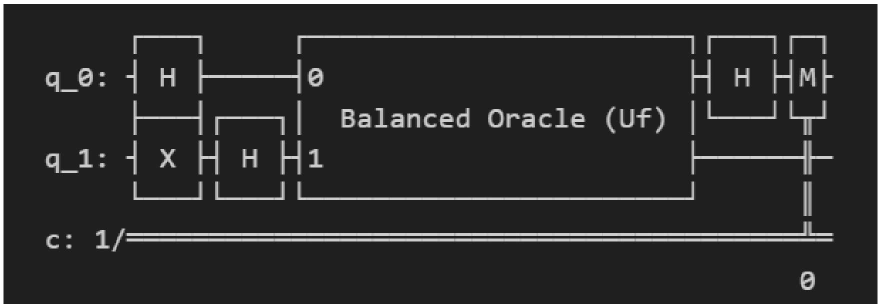

Figure 1: Quantum circuit for the Deutsch Algorithm.

3.8 Results

The simulation results are printed:

```python
print("\n--- Simulation Results ---")
print("Measurement counts:", counts)
print("--------------------------\n")

print("--- Interpretation ---")
if '1' in counts:
    print("The result is '1', which indicates the function is BALANCED.")
else:
    print("The result is '0', which indicates the function is CONSTANT.")
```

The output is `{'1': 4000}`, indicating a balanced function.

4 Results and Discussion

The simulation consistently measures ’1’, confirming the oracle is balanced, as expected from the CNOT-based oracle. The Deutsch Algorithm efficiently distinguishes the function type with one query, showcasing quantum parallelism.