Quantum Programming Project

Khaled Ammoura

Fall 2024-25

Github Repository: https://github.com/Khaled1621/Khaled-Ammoura-Quantum-Programming-Project.git

Problem 1: Order of Quantum Gates

A. Benefit of the Exercise:

Investigate how the order of applying quantum gates affects the final output of a quantum circuit. Specifically, by comparing the outcomes of two circuits that apply the same two rotation gates but in different orders.

B. Tasks:

1. Initialization:
   - $\theta_1: \pi/3$
   - $\theta_2: \pi/4$
2. Circuits:
   - Circuit 1: 0: ──RX(1.05)──RY(0.79)─┤
   - Circuit 2: 0: ──RY(0.79)──RX(1.05)─┤
3. Expectation Values of the Pauli-X observable:
   a. Circuit 1 - Expectation Value of Pauli-X: 0.3536
   b. Circuit 2 - Expectation Value of Pauli-X: 0.7071
4. Probability Distributions of the Measurement Outcomes:
   a. Circuit 1 - Probability |0>: 0.6768, Probability |1>: 0.3232
   b. Circuit 2 - Probability |0>: 0.6768, Probability |1>: 0.3232
5. Absolute Difference between the Pauli-X Expectation Values of the Two Circuits:
   a. Absolute Difference of Pauli-X: 0.3536
6. Expectation Values of the Pauli-Z observable:
   a. Circuit 1 - Expectation Value of Pauli-Z: 0.3536
   b. Circuit 2 - Expectation Value of Pauli-Z: 0.3536

C. Questions:

1. First, we need to note that the expectation value of Pauli-X observable is determined by the state's alignment with the X-axis on the Bloch sphere, meaning Pauli-X measures the projection of the final state along the X-axis. Intuitively, someone would say that we should have the same expectation value in both circuits. For example, let's take a classical example, suppose I have a ball that I should move only in two routines, either up then right, or right then up. Classically, someone would say that immaterial of which routine we take the ball will end at the same place, which is correct, but in classical scenarios, not in quantum mechanics. Quantum rotations don't commute, meaning the order of the applied unitary operations matters in quantum mechanics.
   Going back to the question, assuming for the sack of clarification that theta1 = π/3 and theta2 = π/4, we observed that the expectation value of Pauli-X observable for circuit-1

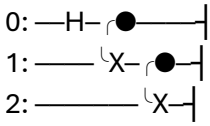
is: 0.3536 and that of circuit-2 is: 0.7071. This means that circuit-2 has a stronger alignment with the X-axis.

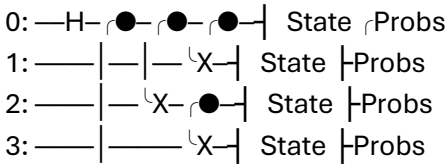
2. Both circuits produce the same ⟨Z⟩ values. This is because the gate combinations in both circuits generate states with the same symmetry along the Z-axis., i.e. absolute difference = 0. Circuit-1: 0.3536 and Circuit-2: 0.3536, there absolute difference = 0, as expected.
3. No. The order in which RX and RY are applied significantly affects ⟨X⟩, but didn't affect that of ⟨Z⟩. Since, for ⟨X⟩ it results in different alignments with the X-axis, however for ⟨Z⟩ it results in a symmetric state indifferent of the order.
4. Someone would notice that the probability of the computation basis (i.e., |0> and |1>) is the same for the 4 circuits (2 for ⟨X⟩, 2 for ⟨Z⟩), because the probability of these gates depends on their projection on the Z-axis, and since there were no rotations along that axis (only on X-axis and Y-axis),the effect of these rotations do not directly affect the state's alignment with the Z-axis in a way that changes the probabilities in the computational basis. Therefore, their probabilities are expected to be the same, and that is what we observed. Circuit 1-2-3-4: P(|0>) = 0.6768 and P(|1>) = 0.3232.

Problem 2: GHZ State Creation and Optimization

A. Benefit of the Exercise:

   Emphasize the importance of circuit optimization by minimizing depth and gate count, fostering skills in efficient quantum circuit design.

B. Tasks:
   1. Design and Implement the Correct Circuit to Produce a 3-qubit GHZ State:
      Generated Quantum Circuits:
      

      Number of Gates = 3
      Depth = 3
   2. Verify the State Vector and Fidelity to Confirm Correctness:
      Generated GHZ State Vector:
      [0.70710678+0.j 0. +0.j 0. +0.j 0. +0.j
      0. +0.j 0. +0.j 0. +0.j 0.70710678+0.j]
      Fidelity with Target GHZ State: 0.9999999999999996
   3. Measure the Expectation Values:
      Expectation Values: [0.0, 0.0, 0.0]
   4. Calculate the Probability Distribution:
      Measurement Probabilities (|000>, ..., |111>): [0.5 0. 0. 0. 0. 0. 0. 0.5]
   5. Design and Implement the Correct Circuit to Produce a 5-qubit GHZ State:
      

4: ——└X————┤ State └Probs
Number of Gates = 5
Depth = 4

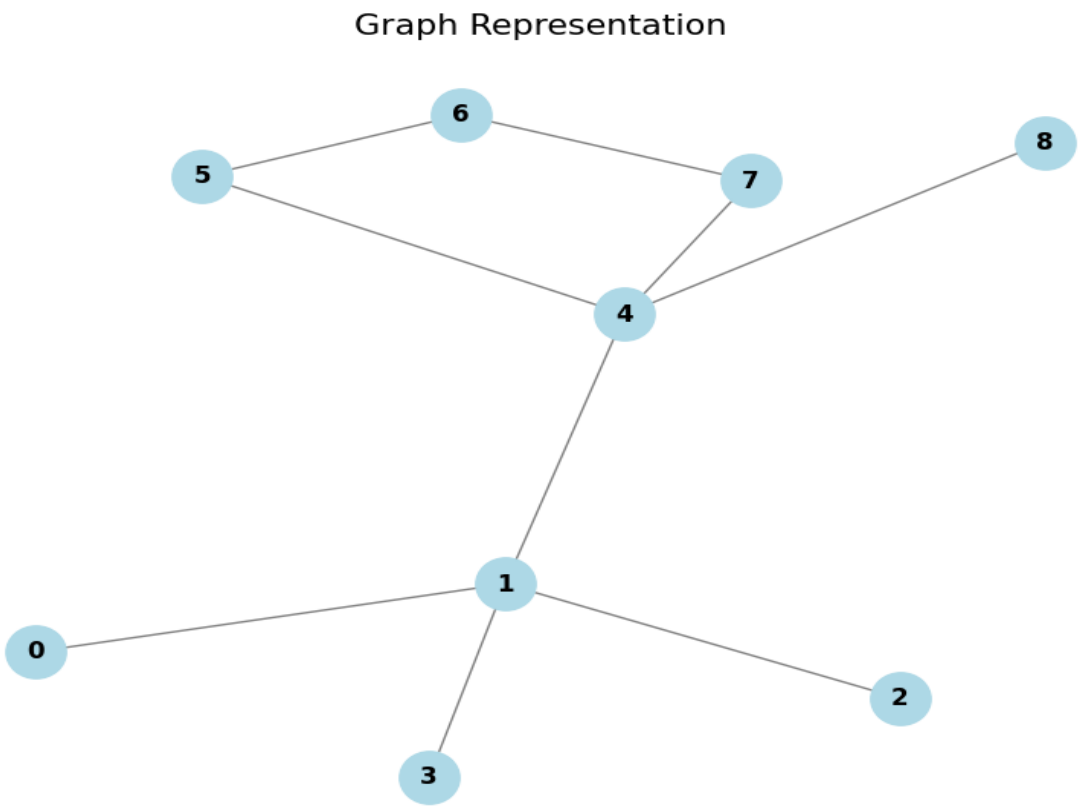
6. Verify the State Vector and Fidelity to Confirm Correctness:
   [0.70710678+0.j 0. +0.j 0. +0.j 0. +0.j
   0. +0.j 0. +0.j 0. +0.j 0. +0.j
   0. +0.j 0. +0.j 0. +0.j 0. +0.j
   0. +0.j 0. +0.j 0. +0.j 0. +0.j
   0. +0.j 0. +0.j 0. +0.j 0. +0.j
   0. +0.j 0. +0.j 0. +0.j 0. +0.j
   0. +0.j 0. +0.j 0. +0.j 0. +0.j
   0. +0.j 0. +0.j 0. +0.j 0.70710678+0.j]
   Fidelity with Target GHZ State: 0.9999999999999996
7. Measure the Expectation Values:
   Expectation Values: [0.0, 0.0, 0.0, 0.0, 0.0]
8. Measure the Probability Distribution:
   Measurement Probabilities (|00000>, ..., |11111>): [0.5 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.5]
9. Comparison:
   a. GHZ-3:
      i. Number of Gates: 1 Hadamard + 2 CNOT → 3 Gates
      ii. Depth of the Circuit: 3
   b. GHZ-5:
      i. Number of Gates: 1 Hadamard + 4 CNOT → 5 Gates
      ii. Depth of the Circuit: 4

   We can see that the number of Gates increases linearly as the number of qubits increases. However, it is not the case in the depth since the optimized GHZ-5 have a depth 4 not 5.

Problem 3: Counting SWAP Gates Needed for a CNOT Gate

A. Benefit of the Exercise:
   Understand how limited connectivity in quantum hardware affects circuit design and how to optimize gate operations by inserting SWAP gates to enable CNOT operations.
B. Tasks:

1. Visualize the graph:



2. Use Breadth-First Search (BFS)
   Since it guarantees the shortest path in an unweighted graph. If we find the shortest path between the control qubit and the target qubit we just subtract from their distance -1 and then multiply by 2 to account for the target qubit going back to its original position.
3. Test on CNOT(0,4):
   Minimum number of SWAP gates required: 2

Problem 4: Deutsch's Algorithm Implementation

A. Benefit of the Exercise:
   This exercise demonstrates the power of quantum parallelism and how quantum algorithms can solve problems more efficiently than classical ones. Specifically, Deutsch's algorithm determines if a function is constant or balanced with only one query to the oracle, while a classical algorithm would require two queries. It helps understand the fundamental concepts like quantum superposition, interference, and oracles, which are essential building blocks for more complex quantum algorithms.
B. Tasks:
   1. Implement the oracle which determines whether a function is constant or balanced based on measurement outcome:
      a. Constant Function: For example,
         i. constant_f = lambda x: 0 # f(x)=0

b. Balanced Function: For example,
    i. balanced_f = lambda x: x # f(x)=x

2. Design and Implement Deutsh's Circuit:
    a. Constant Circuit:
    0: —H—H—┤ <Z>
    1: —X—H—┤
    Apply Identity
    b. Balanced Circuit:
    0: ——H———╭●——H—┤ <Z>
    1: ——X——H—╰X———┤
    Apply CNOT0,1

3. Expectation Values of the Pauli-Z observable:
    a. Constant Circuit:
    Measurement result for constant function f(x) = 0: 0.9999999999999996→0
    b. Balanced Circuit:
    Measurement result for balanced function f(x) = x: -0.9999999999999996→1

Problem 5: Deutsch-Jozsa Algorithm Implementation

A. Benefit of the Exercise:
This exercise demonstrates the power of quantum parallelism and how the Deutsch-Jozsa algorithm efficiently determines whether a function is constant or balanced. While a classical algorithm would require up to $2^{n-1}+1$ queries to determine this, the Deutsch-Jozsa algorithm accomplishes it with a single query to the oracle. It highlights the role of quantum superposition, where all possible inputs are evaluated simultaneously, and quantum interference, which amplifies the correct outcome. Understanding these concepts, including the function-encoding oracles and the Hadamard gates that enable superposition, lays a foundation for more advanced quantum algorithms and showcases the potential speedups offered by quantum computing.

B. Tasks:
1. Implement the oracle which determines whether a function is constant or balanced based on measurement outcome.
    a. Constant Function:
    constant_f = lambda x: 0 # f(x)=0
    b. Balanced Function:
    balanced_f = lambda x: x # f(x)=x
2. Description of Each Function and Oracle Encoding:
    a. Constant Function f(x)=0:
        i. Description: This function always returns 0 regardless of the input.
        ii. Oracle Encoding: The oracle for this function applies no operations since f(x)=0 for all x. Therefore, no CNOT gates are applied to the auxiliary qubit.
        iii. Quantum Circuit: The input qubits remain unaffected after the oracle, and the Hadamard gates ensure constructive interference, resulting in a positive measurement outcome.
    b. Balanced Function f(x)=x:

i. Description: This function returns 0 for half the inputs and 1 for the other half. For two qubits, it maps 0 to 0 and 1 to 1.
ii. Oracle Encoding: The oracle applies a CNOT gate controlled by the input qubits, targeting the auxiliary qubit, flipping it when f(x)=1. This creates interference patterns that affect the measurement outcomes.
iii. Quantum Circuit: The Hadamard gates transform the qubit states such that destructive interference leads to a negative measurement outcome for at least one qubit.

3. Output of the Deutsch-Jozsa Algorithm:
   a. Constant Function:
      i. Circuit:
      0: —H—H—┤ <Z>
      1: —H—H—┤ <Z>
      2: —X—H—┤
      ii. Expectation Values of the Pauli-Z observable:
      Measurement result for constant function f(x) = 0:
      [0.9999999999999992, 0.9999999999999992]. This corresponds to a positive expectation value for both qubits, indicating that the function is constant.
   b. Balanced Function:
      i. Circuit:
      0: ──H──H──────────┤ <Z>
      1: ──H──────╭●──H──┤ <Z>
      2: ──X──H──╰X──────┤
      ii. Expectation Values of the Pauli-Z observable:
      Measurement result for balanced function f(x) = x:
      [0.9999999999999992, -0.9999999999999992]. The second qubit shows a negative expectation value due to destructive interference, indicating that the function is balanced.
4. Explanation of Results:
   a. Constant Function:
      The Hadamard gates create superpositions, but since the oracle does not modify the states (no flips occur), constructive interference leads to positive measurement results.
   b. Balanced Function:
      The CNOT gate in the oracle flips the auxiliary qubit based on the input, causing destructive interference in the Hadamard transform of the input qubits. This leads to at least one negative measurement result, distinguishing it as a balanced function.

C. Questions:
1. Hadamard gates in the Deutsch and Deutsch-Jozsa algorithms are essential for creating quantum superposition and enabling interference. Initially, they transform the input qubits from the state $|0\rangle$ to a superposition of all possible states, allowing the algorithm to evaluate the function f(x) for all inputs simultaneously (quantum parallelism). After the oracle processes the function, Hadamard gates are applied again to interfere with the qubit states. This interference pattern helps distinguish

between constant and balanced functions based on the measurement outcomes, enabling the algorithm to achieve results with significantly fewer queries compared to classical approaches.

2. An oracle is constant if it encodes a function $f: \{0,1\}^n \longrightarrow \{0,1\}$ that returns the same output (either always 0 or always 1) for all possible inputs. Conversely, an oracle is balanced if the function f returns 0 for exactly half of the inputs and 1 for the other half.
    a. Constant Functions:
        i. f(x)=0
        ii. f(x)=1
    b. Balanced Functions: assume two bits function
        i. f(x)=x_1
        This function returns the first bit of the input x.
        ii. f(x)=$x_1 \oplus x_2$
        This function returns the parity (XOR) of the bits in x.

Problem 6* (Bonus): Quantum Superdense Coding Using Bell Pairs