

Date:09/09/25

## TASK 8: Grover's algorithm for a 3-qubits database

Aim: To implement Grover's quantum search algorithm for a 3-qubit search space (8 items) using Cirq, and demonstrate that the marked item (target state) can be found with high probability after the optimal number of iterations.

### 1 Mathematical Model of the Grover's algorithm for a 3-qubits database

- Database size: Search space  $N = 2^3 = 8$  elements represented by 3-qubit basis states  $\{|000\rangle, |001\rangle, \dots, |111\rangle\}$ .

Target: A basis state  $|w\rangle$ , here chosen as  $|101\rangle$ .

- Initial State: Equal superposition  $|S\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle = \frac{1}{\sqrt{8}} \sum_{x=0}^7 |x\rangle$  created by applying Hadamard gates on all 3 qubits.

- Oracle:  $f(x) = 1$  if  $x = x'$  (the “marked” target), 0 otherwise.

$$O|x\rangle = -|x\rangle \text{ if } x = w; \text{ else } O|x\rangle = |x\rangle.$$

- Diffusion operator: Inversion about the mean, realized by Hadamard gates, X gates, multi-controlled Z, and reversing these gates.

$$D = 2|s\rangle\langle s| - I$$

- Grover operator:  $G = D \cdot O$ .

- Grover Iterations: Apply oracle + diffusion operator approximately  $r = \frac{\pi}{4} \sqrt{\frac{N}{M}}$  times to amplify the amplitude of the marked state.

For  $N = 8$ ,  $M=1$ ,  $r = 2$ .

- Measurement: Measurement of all qubits follows iterations, producing bit strings with probability concentrated on the marked state.

### 2 Algorithm - Grover's algorithm for a 3-qubits database

1. Initialize 3 qubits to  $|0\rangle$ .
2. Create uniform superposition by Hadamard gates  $H \otimes 3$ .

3. Repeat  $k = 2$  times:
  - o Apply oracle marking the target bit string with phase flip on the marked state.
  - o Apply diffusion operator (inversion about the mean).
4. Measure the qubits to obtain results peaked at the target state with high probability.

### 3 Program

```

import cirq import numpy as
np import matplotlib.pyplot as
plt

def grover_3_qubit(target_binary):
    qubits = [cirq.GridQubit(0, i) for i in range(3)]
    circuit = cirq.Circuit()

    # Initialize superposition
    circuit.append(cirq.H.on_each(*qubits))
    # Removed: circuit.append(cirq.Barrier(*qubits)) # Barrier after initialization

    # Number of Grover iterations
    N = 2 ** 3
    iterations = int(np.floor(np.pi/4 * np.sqrt(N)))

    for iteration in range(iterations):
        # Oracle
        apply_oracle(circuit, qubits, target_binary)
        # Removed: circuit.append(cirq.Barrier(*qubits)) # Barrier after oracle

        # Diffusion
        apply_diffusion(circuit, qubits)
        # Removed: circuit.append(cirq.Barrier(*qubits)) # Barrier after diffusion

    # Measurement
    circuit.append(cirq.measure(*qubits, key='result'))

    return circuit, qubits

def apply_oracle(circuit, qubits, target_binary):
    # Apply X gates where target bit is 0
    for i, bit in enumerate(target_binary):
        if bit == '0':
            circuit.append(cirq.X(qubits[i]))

    # Multi-controlled Z using H and CCX
    circuit.append(cirq.H(qubits[-1]))
    circuit.append(cirq.CCX(qubits[0], qubits[1], qubits[2]))
    circuit.append(cirq.H(qubits[-1]))

```

```

# Undo X gates    for i, bit in
enumerate(target_binary):      if bit
== '0':
    circuit.append(cirq.X(qubits[i]))


def apply_diffusion(circuit, qubits):
    circuit.append(cirq.H.on_each(*qubits))
    circuit.append(cirq.X.on_each(*qubits))
    circuit.append(cirq.H(qubits[-1]))
    circuit.append(cirq.CCX(qubits[0], qubits[1], qubits[2]))
    circuit.append(cirq.H(qubits[-1]))
    circuit.append(cirq.X.on_each(*qubits))
    circuit.append(cirq.H.on_each(*qubits))



def analyze_results(counts, target):
    total = sum(counts.values())
    success = counts.get(int(target, 2), 0)
    success_rate = success / total * 100

    print(f'Measurement results for target |{target}>:')
    for state in range(8):
        bitstr = format(state, '03b')
        count = counts.get(state, 0)
        pct = count / total * 100
        marker = "<-- Target" if bitstr == target else ""
        print(f"State |{bitstr}>: {count} times ({pct:.2f}%) {marker}")

    print(f"\nSuccess rate: {success_rate:.2f}% (optimal ~94% after 2 iterations)")


states = [format(i, '03b') for i in range(8)]    values
= [counts.get(i, 0) for i in range(8)]    colors = ['red' if
s == target else 'blue' for s in states]

plt.bar(states, values, color=colors)    plt.title(f"Grover's
Algorithm Results (Target: |{target}>)")
plt.xlabel("States")    plt.ylabel("Counts")    plt.show()


if __name__ == "__main__":
    target =
    "101"
    circuit, qubits =
    grover_3_qubit(target)

    print("Circuit diagram:")
    print(circuit)

```

```

simulator = cirq.Simulator()    result =
simulator.run(circuit, repetitions=1000)    counts
= result.histogram(key='result')

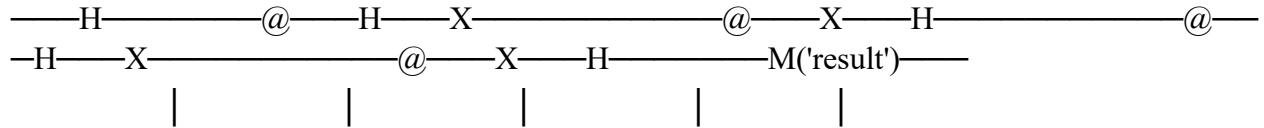
analyze_results(counts, target)

```

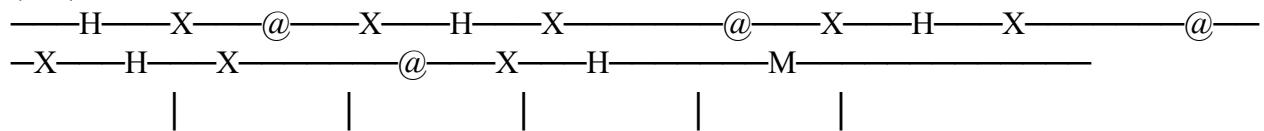
Output:

Circuit diagram:

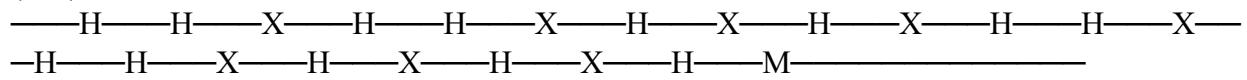
(0, 0):



(0, 1):



(0, 2):



Measurement results for target  $|101\rangle$ :

State  $|000\rangle$ : 6 times (0.60%)

State  $|001\rangle$ : 7 times (0.70%)

State  $|010\rangle$ : 11 times (1.10%)

State  $|011\rangle$ : 9 times (0.90%)

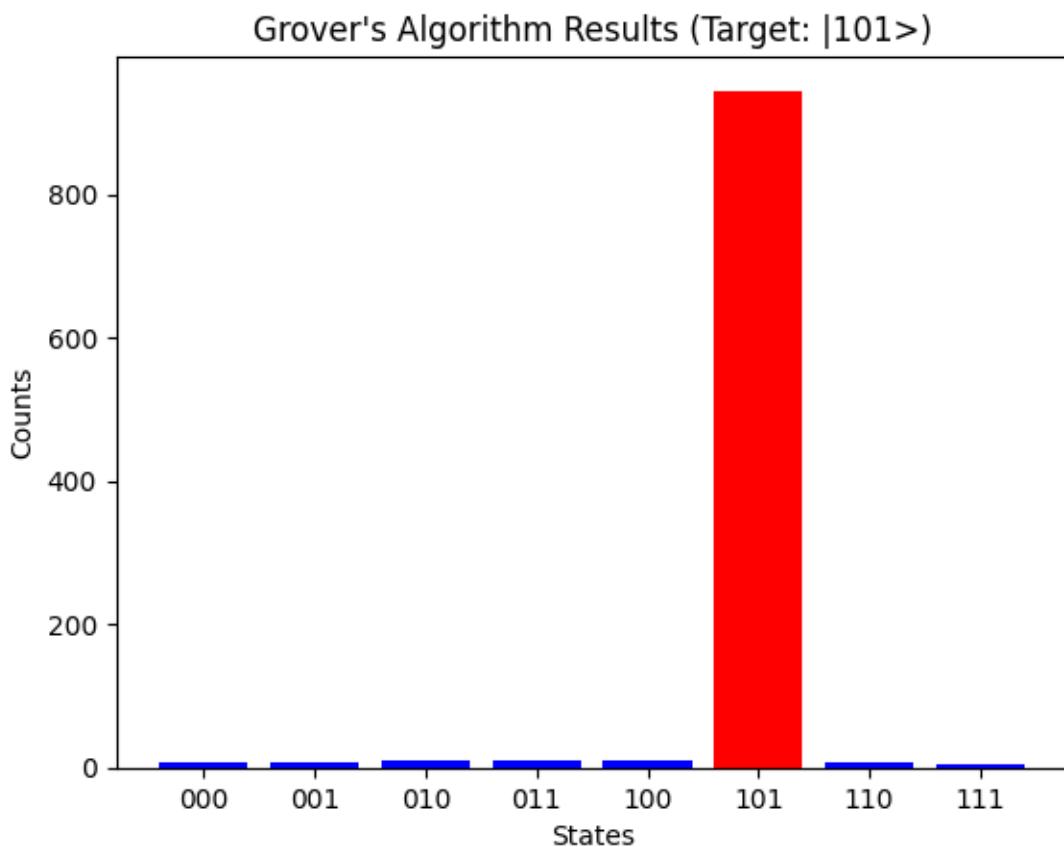
State  $|100\rangle$ : 9 times (0.90%)

State  $|101\rangle$ : 945 times (94.50%) <-- Target

State  $|110\rangle$ : 8 times (0.80%)

State  $|111\rangle$ : 5 times (0.50%)

Success rate: 94.50% (optimal ~94% after 2 iterations)



#### 4 Result

The successful implementation of Grover's algorithm for a 3-qubit database validates the theoretical foundations of quantum search algorithms and provides a practical demonstration of quantum computing potential for solving problems more efficiently than classical computers