2025 / Spring Semester

Topics on Quantum Computing

Lecture Note – Grover's Search Algorithm

Junghee Ryu

KISTI

Grover's Search Algorithm (GSA)

A "quantum" search protocol

- The Grover's Search Algorithm (GSA) is a well-known search protocol that can, in principle, beat the classical searching in terms of the workload complexity.
  - An oracle-based algorithm (oracle - will be covered in next slides)
- The problem is defined by "searching for an item on a list of N items, given an oracle-access function f(x): f(x) = 1 if x is the item we want, and 0 otherwise".
  - To solve this black-box search problem, the most primitive classical approach would require a total of N trials in the worst case.
  - GSA needs up to ~ $N^{1/2}$ queries to the list for finding the answer.
- Let's check how GSA works in next sides.

Grover's Search Algorithm (GSA)

A "quantum" search protocol

- GSA can be broken down into the following steps.
  - (STEP 1) Prepare the initial state
  - (STEP 2) Implement the oracle circuit
  - (STEP 3) Apply the Grover diffusion operator
  - (STEP 4) Repeat STEP 2 & 3 approximately up to ~ $N^{1/2}$ times
  - (STEP 5) Get (measure) the result
- In this class, we are going to implement a search protocol for an n-bit string item using a quantum circuit based on GSA
  - n-bit string: the classical brute-force search would require up to $N = 2^n$ trials to get the desired solution.

Grover's Search Algorithm (GSA)

A "quantum" search protocol

- (STEP 1) Preparation of the initial state
  - To perform the search, we are going to create an n-dimensional system, which has N = 2^n states that are represented with N binary numbers. Let's say these binary numbers are $x_0, x_1, \dots, x_{N-2}, x_{N-1}$.
  - We first need to initialize the system in the uniform superposition over all states - i.e., the amplitudes associated with each of the N basis states are equal.

$$|s\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle$$

  - $|s\rangle$ can be obtained by applying a Hadamard gate to all the wires of an n-qubit circuit

Grover’s Search Algorithm (GSA)

A “quantum” search protocol

- ❑ (STEP 1) Preparation of the initial state

```
import matplotlib.pyplot as plt
import pennylane as qml
import numpy as np

NUM_QUBITS = 2
dev = qml.device("default.qubit", wires=NUM_QUBITS) # 2-qubit circuit in emulator device
wires = list(range(NUM_QUBITS))                      # [0, 1]

def equal_superposition(wires):
    for wire in wires:
        qml.Hadamard(wires=wire)

# continued in the next slide
```

Grover’s Search Algorithm (GSA)

A “quantum” search protocol

❑ (STEP 1) Preparation of the initial state

- The `Snapshot` operation saves the internal execution state of the quantum function at a specific point in the execution pipeline.
- This is a pseudo-operation with no effect on the state. Arbitrary measurements are supported in snapshots via the keyword argument `measurement`

https://docs.pennylane.ai/en/stable/code/api/pennylane.Snapshot.html

```
# continued from previous slide

@qml.qnode(dev)
def circuit():
    qml.Snapshot("Initial state")
    equal_superposition(wires)
    qml.Snapshot("After applying the Hadamard gates")
    return qml.probs(wires=wires)
    # Probability of finding a computational basis state on the wires

results = qml.snapshots(circuit)()

for k, result in results.items():
   print(f"{k}: {result}")
```

```
Initial state: [1.+0.j 0.+0.j 0.+0.j 0.+0.j]
After applying the Hadamard gates: [0.5+0.j 0.5+0.j 0.5+0.j 0.5+0.j]
execution_results: [0.25 0.25 0.25 0.25]
```

Grover’s Search Algorithm (GSA)

A “quantum” search protocol

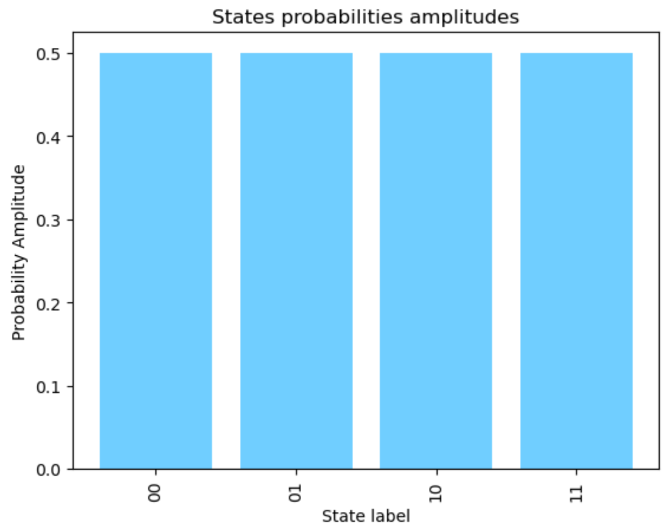
❑ (STEP 1) Preparation of the initial state

```
# continued from previous slide

y = np.real(results["After applying the Hadamard gates"])
bit_strings = [f"{x:0{NUM_QUBITS}b}" for x in range(len(y))]

plt.bar(bit_strings, y, color = "#70CEFF")

plt.xticks(rotation="vertical")
plt.xlabel("State label")
plt.ylabel("Probability Amplitude")
plt.title("States probabilities amplitudes")
plt.show()
```



Grover's Search Algorithm (GSA)

A "quantum" search protocol

- ❑ (STEP 2) Implementation of the oracle circuit
  - Let's assume for now that only one index satisfies $f(x) = 1$. We call this index ω.
  - We define the oracle access operator U_ω such that

$$\begin{cases} U_\omega|x\rangle = -|x\rangle & \text{for } x = \omega, \text{ that is, } f(x) = 1, \\ U_\omega|x\rangle = |x\rangle & \text{for } x \neq \omega, \text{ that is, } f(x) = 0, \end{cases}$$

  - U_ω acts by flipping the phase of the solution state while keeping the remaining states untouched. In other words, the unitary U_ω can be seen as **a reflection around the set of orthogonal states to $|\omega\rangle$**.

$$U_\omega = \mathbb{I} - 2|\omega\rangle\langle\omega| \quad \Rightarrow \quad \begin{aligned} U_\omega|x\rangle &= (I - 2|\omega\rangle\langle\omega|)|x\rangle \\ &= |x\rangle - 2|\omega\rangle\langle\omega|x\rangle \end{aligned}$$

Grover's Search Algorithm (GSA)

A "quantum" search protocol

- (STEP 2) Implementation of the oracle circuit
  - Let's assume for now that only one index satisfies $f(x) = 1$. We call this index ω.
  - We define the oracle access operator U_ω such that

$$\begin{cases} U_\omega|x\rangle = -|x\rangle & \text{for } x = \omega, \text{ that is, } f(x) = 1, \\ U_\omega|x\rangle = |x\rangle & \text{for } x \neq \omega, \text{ that is, } f(x) = 0, \end{cases}$$

  - U_ω acts by flipping the phase of the solution state while keeping the remaining states untouched. In other words, the unitary U_ω can be seen as **a reflection around the set of orthogonal states to $|\omega\rangle$**.

$$U_\omega = \mathbb{I} - 2|\omega\rangle\langle\omega| \quad \Rightarrow$$

$$U_\omega|x\rangle = (I - 2|\omega\rangle\langle\omega|)|x\rangle$$
$$= |x\rangle - 2|\omega\rangle\langle\omega|x\rangle$$

Grover's Search Algorithm (GSA)

A "quantum" search protocol

- ❑ (STEP 2) Implementation of the oracle circuit
  - The oracle can be easily secured with the `FlipSign` in Pennylane.

```
dev = qml.device("default.qubit", wires=NUM_QUBITS)
@qml.qnode(dev)
def circuit():
    qml.Snapshot("Initial state |00>")
    qml.FlipSign([0,0], wires=wires) # Flipping the marked state
    qml.Snapshot("After flipping it")
    return qml.state()
results = qml.snapshots(circuit)()

for k, result in results.items():
   print(f"{k}: {result}")
```

```
Initial state |00>: [1.+0.j 0.+0.j 0.+0.j 0.+0.j]
After flipping it: [-1.+0.j 0.+0.j 0.+0.j 0.+0.j]
execution_results: [-1.+0.j 0.+0.j 0.+0.j 0.+0.j]
```

Grover's Search Algorithm (GSA)

A "quantum" search protocol

❑ (STEP 2) Implementation of the oracle circuit

```
# The code continued from previous slide 7

omega = np.zeros(NUM_QUBITS)

def oracle(wires, omega):
    qml.FlipSign(omega, wires=wires)

@qml.qnode(dev)
def circuit2():
    equal_superposition(wires)
    qml.Snapshot("Before querying the Oracle")
    oracle(wires, omega)
    qml.Snapshot("After querying the Oracle")
    return qml.probs(wires=wires)
(...)
```

```
Before querying the Oracle: [0.5+0.j 0.5+0.j 0.5+0.j 0.5+0.j]
After querying the Oracle: [-0.5+0.j 0.5+0.j 0.5+0.j 0.5+0.j]
execution_results: [0.25 0.25 0.25 0.25]
```

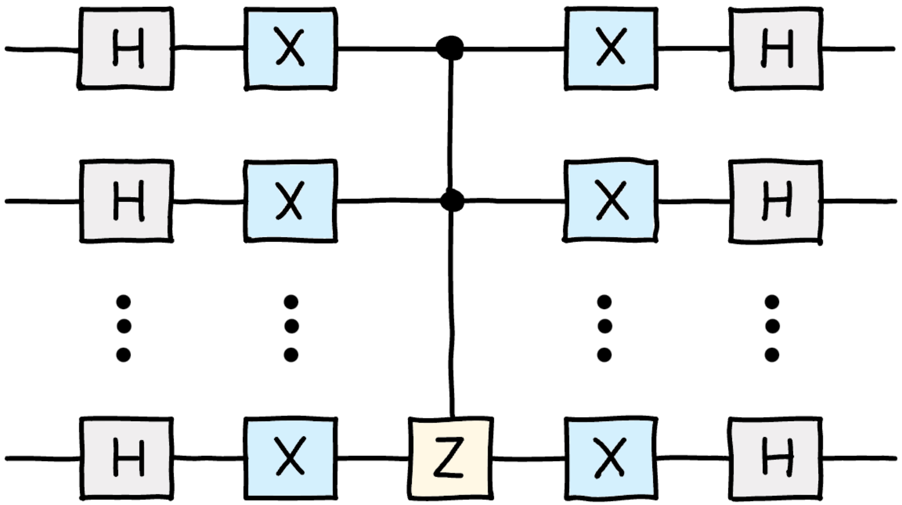
```
(...)
results = qml.snapshots(circuit2)()

for k, result in results.items():
print(f"{k}: {result}")
```

Grover's Search Algorithm (GSA)

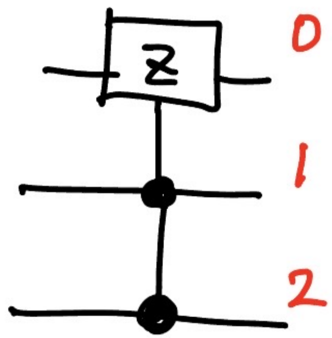
A "quantum" search protocol

- ❑ (STEP 3) Diffusion operator
  - Finding the solution needs an additional step since the probability of measuring any of the states remains equally distributed - the Grover diffusion operator that can be represented with the following circuit in a general manner.



- The multi-controlled Z gate : `qml.ctrl(…)`

```
qml.ctrl(qml.PauliZ,[1,2],control_values=[1,1])(wires=0)
```



Grover’s Search Algorithm (GSA)

A “quantum” search protocol

❑ (STEP 3) Diffusion operator

- Let’s see what happens when we apply the diffusion operator.

```
# The code continued from previous slide 11

def diffusion():
    for wire in range(NUM_QUBITS):
        qml.Hadamard(wires=wire)
        qml.PauliX(wires=wire)

        qml.ctrl(qml.PauliZ, range(NUM_QUBITS-1), control_values=np.ones(NUM_QUBITS-1))(
wires=NUM_QUBITS-1)


    for wire in range(NUM_QUBITS):
        qml.PauliX(wires=wire)
        qml.Hadamard(wires=wire)

# continued in the next slide
```

Grover's Search Algorithm (GSA)

A "quantum" search protocol

- (STEP 3) Diffusion operator
  - Let's see what happens when we apply the diffusion operator.

```
# continued from previous slide

@qml.qnode(dev)
def test():
    equal_superposition(wires)
    qml.Snapshot("state 1")
    oracle(wires,omega)
    qml.Snapshot("state 2")
    diffusion()
    qml.Snapshot("state 3")
    return qml.state()

results = qml.snapshots(test)()
(...)
```

```
(...)
for k, result in results.items():
    print(f"{k}: {result}")

# continued in the next slide
```

```
state 1: [0.5+0.j 0.5+0.j 0.5+0.j 0.5+0.j]
state 2: [-0.5+0.j  0.5+0.j  0.5+0.j 0.5+0.j]
state 3: [-1.+0.j  0.+0.j  0.+0.j 0.+0.j]
execution_results: [-1.+0.j  0.+0.j  0.+0.j 0.+0.j]
```

What do you see from the result?
→ Amplification

Grover's Search Algorithm (GSA)

A "quantum" search protocol

- ❑ (STEP 4) Repeat STEP 3 & STEP 4
  - ▪ The complete form of GSA.

```
# continued from previous slide
@qml.qnode(dev)
def myGSA(Niter):
    equal_superposition(wires)
    for i in range(Niter):
       oracle(wires,omega)
       diffusion()
    return qml.probs()

result = np.array([])
for i in range(10):
   temp = myGSA(i)
   result = np.append(result, temp[0])
print(result)
```

```
[0.25 1. 0.25 0.25 1. 0.25 0.25 1. 0.25 0.25]
```

What appens if we blindly repeat this?

Let's set the iteration # to
`(round) pi/4*sqrt(NUM_QUBITS)`

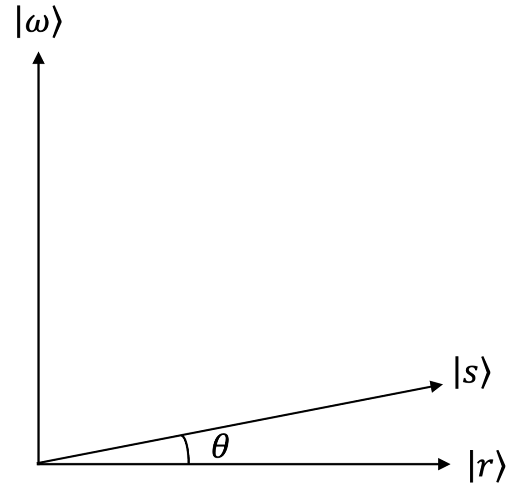
* (round) pi/4*sqrt(2) = 1

Grover's Search Algorithm (GSA)

A "quantum" search protocol

- ❑ (STEP 4) Repeat STEP 2 & STEP 3
  - Let's note the diffusion operator with U_D.
  - The oracle operator: U_ω (slide 8), the equally superposed state: $|s\rangle$ (slide 4)

$$
\begin{aligned}
|s\rangle &\coloneqq H^{\otimes n}|0\rangle^{\otimes n} \quad \textbf{(STEP 1)} \\
&= \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)\otimes\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)\otimes\cdots\otimes\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \\
&= \frac{1}{\sqrt{N}}\sum_{x\in\{0,1\}^n}|x\rangle \\
&= \frac{1}{\sqrt{N}}|\omega\rangle + \frac{\sqrt{N-1}}{\sqrt{N}}\frac{1}{\sqrt{N-1}}\sum_{x\neq\omega}|x\rangle
\end{aligned}
$$



Let $\theta = \sin^{-1}\left(\frac{1}{\sqrt{N}}\right)$ and $|r\rangle \coloneqq \frac{1}{\sqrt{N-1}}\sum_{x\neq\omega}|x\rangle$. Then $|s\rangle = \sin\theta\,|\omega\rangle + \cos\theta\,|r\rangle$.

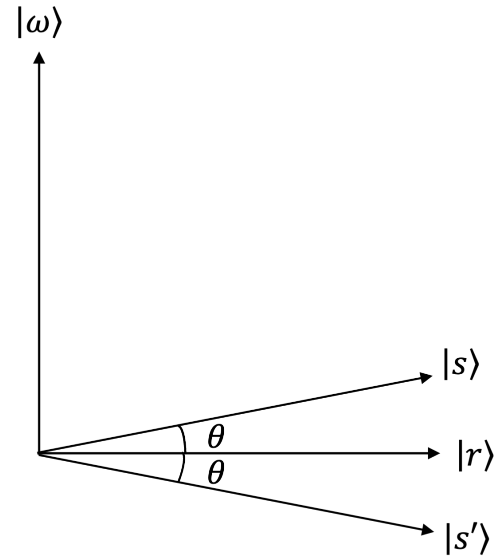
Grover's Search Algorithm (GSA)

A "quantum" search protocol

- ❑ (STEP 4) Repeat STEP 2 & STEP 3
  - Let's note the diffusion operator with U_D.
  - The oracle operator: U_ω (slide 8), the equally superposed state: $|s\rangle$ (slide 4)

$$U_\omega|s\rangle = (I - |\omega\rangle\langle\omega|)(\sin\theta|\omega\rangle + \cos|r\rangle)$$

(STEP 2) $$= -\sin\theta|\omega\rangle + \cos\theta|r\rangle = |s'\rangle$$



$$U_D|s'\rangle = (2|s\rangle\langle s| - I)|s'\rangle$$

$$= \sin 3\theta|\omega\rangle + \cos 3\theta|r\rangle$$

Grover's Search Algorithm (GSA)

A "quantum" search protocol

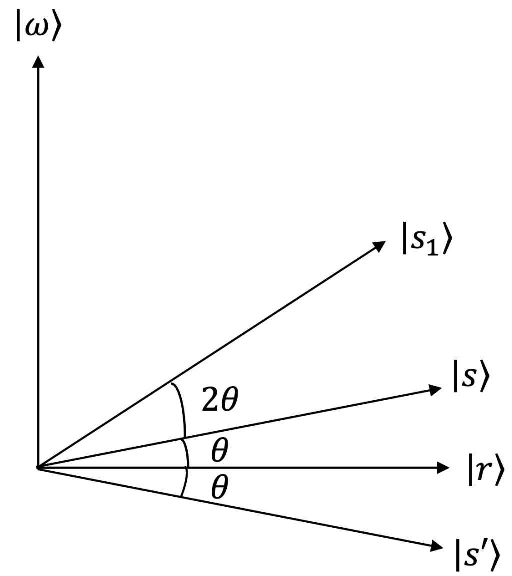
- (STEP 4) Repeat STEP 2 & STEP 3
  - Let's note the diffusion operator with U_D.
  - The oracle operator: U_ω (slide 8), the equally superposed state: $|s\rangle$ (slide 4)

$$U_\omega |s\rangle = (I - |\omega\rangle\langle\omega|)(\sin\theta|\omega\rangle + \cos|r\rangle)$$
$$= -\sin\theta|\omega\rangle + \cos\theta|r\rangle = |s'\rangle$$

Again: What appens if we blindly repeat STEP 2 and 3?

$$U_D|s'\rangle = (2|s\rangle\langle s| - I)|s'\rangle$$

(STEP 3) $= \sin 3\theta|\omega\rangle + \cos 3\theta|r\rangle = |s_1\rangle$



Quantum Phase Estimation (QPE)

Coding Practice

- Write your own GSA using the contents we have discussed so far
  - No skeleton code is given
  - Circuit size (`NUM_QUBITS`) and Oracle-accessing state must be controllable by users
  - The code iterates STEP 2 & STEP 3 by N times, where N is fixed to `(round)pi/4*sqrt(NUM_QUBITS)`
- Check the result with following conditions to make sure your code works fine
  - `Access state = 01010, NUM_QUBITS = 5`
  - `Access state = 0110100, NUM_QUBITS = 7`
  - `Access state = 0101010101, NUM_QUBITS = 10`