COMS 4281 - Introduction to Quantum Computing Fall 2026

Practice Worksheet 4 - Quantum algorithms for search and counting

Solutions

Problem 1:

(a) Each Grover iteration uses one phase-oracle call, costing T gates, and one diffusion operation, costing $O(n)$ gates by Problem 2. Thus the overall gate complexity is

$$O\big((T+n)\sqrt{2^n}\big).$$

When $T = \Omega(n)$, this is $O(T\sqrt{2^n})$.

(b) Database search requires an oracle that looks up a record and checks whether it satisfies the desired condition. The query count does not include the cost of implementing that lookup.

For example, if a direct circuit implementation of an unstructured table of N records uses $\Theta(N)$ gates per lookup, then $O(\sqrt{N})$ Grover iterations cost $O(N^{3/2})$ gates, rather than $O(\sqrt{N})$. A different memory-access model can change the lookup cost. The title alone does not establish an efficient implementation for an ordinary database.

(c) Functions f that are inexpensive to compute, but for which finding an input with $f(x) = 1$ is difficult. For example, a polynomial-size circuit can check whether a candidate input has a specified hash value. The oracle should be a small computation, not a large table of precomputed answers.

Problem 2: Grover diffusion operator

(a) Since $H|+\rangle = |0\rangle$,

$$H^{\otimes n}|s\rangle = |0^n\rangle, \qquad H^{\otimes n}|s\rangle\langle s|H^{\otimes n} = |0^n\rangle\langle 0^n|.$$

(b) The OR function is 0 only at 0^n. Hence its phase oracle fixes $|0^n\rangle$ and negates every other computational-basis state:

$$O_{\mathrm{OR}} = 2|0^n\rangle\langle 0^n| - I.$$

Conjugating by $H^{\otimes n}$ gives

$$H^{\otimes n} O_{\mathrm{OR}} H^{\otimes n} = 2|s\rangle\langle s| - I = R.$$

(c) A binary tree of two-bit OR gates computes the n-bit OR using $n-1$ gates. Convert it to a reversible circuit, apply the phase, and uncompute the working bits. This uses $O(n)$ elementary gates with the reversible-gate construction from class. The two layers of Hadamards add $2n$ gates, so R has gate complexity $O(n)$.

Problem 3: Analyzing Grover's algorithm with multiple solutions

Assume $0 < M < N$, so both $|\Delta\rangle$ and $|\Gamma\rangle$ are defined. Write $\sin\theta = \sqrt{M/N}$ and $\cos\theta = \sqrt{(N-M)/N}$.

(a) Split the uniform superposition into unmarked and marked inputs:

$$|\psi_0\rangle = \frac{1}{\sqrt{N}} \sum_x |x\rangle = \sqrt{\frac{N-M}{N}}\,|\Delta\rangle + \sqrt{\frac{M}{N}}\,|\Gamma\rangle = \cos\theta\,|\Delta\rangle + \sin\theta\,|\Gamma\rangle\,.$$

(b) In the ordered basis $|\Delta\rangle, |\Gamma\rangle$, the phase oracle and diffusion operator are

$$O_f = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, \qquad R = \begin{pmatrix} 2\cos^2\theta - 1 & 2\sin\theta\cos\theta \\ 2\sin\theta\cos\theta & 2\sin^2\theta - 1 \end{pmatrix} = \begin{pmatrix} \cos 2\theta & \sin 2\theta \\ \sin 2\theta & -\cos 2\theta \end{pmatrix}.$$

One iteration is $G = RO_f$, so

$$G = \begin{pmatrix} \cos 2\theta & -\sin 2\theta \\ \sin 2\theta & \cos 2\theta \end{pmatrix}.$$

Using the angle-addition formulas,

$$G \begin{pmatrix} \cos\theta \\ \sin\theta \end{pmatrix} = \begin{pmatrix} \cos 3\theta \\ \sin 3\theta \end{pmatrix}.$$

Thus $|\psi_1\rangle = \cos(3\theta)\,|\Delta\rangle + \sin(3\theta)\,|\Gamma\rangle$.

(c) More generally, $G(\cos\alpha, \sin\alpha)^T = (\cos(\alpha + 2\theta), \sin(\alpha + 2\theta))^T$. Each iteration adds 2θ to the angle. Starting from angle θ gives

$$|\psi_k\rangle = \cos((2k+1)\theta)\,|\Delta\rangle + \sin((2k+1)\theta)\,|\Gamma\rangle\,.$$

(d) The probability of a particular outcome x is

$$\Pr(x) = \begin{cases} \dfrac{\cos^2((2k+1)\theta)}{N-M}, & f(x) = 0, \\ \dfrac{\sin^2((2k+1)\theta)}{M}, & f(x) = 1. \end{cases}$$

Thus the total success probability is $\sin^2((2k+1)\theta)$, and conditioned on success, the output is uniformly distributed over the M solutions.

(e) Choose a nonnegative integer k nearest to

$$\frac{\pi}{4\theta} - \frac{1}{2}.$$

Then $(2k+1)\theta$ is within θ of $\pi/2$, so success is at least $\cos^2\theta = 1 - M/N$. For $M \ll N$, $\theta \approx \sqrt{M/N}$, giving

$$k \approx \frac{\pi}{4}\sqrt{\frac{N}{M}} - \frac{1}{2} = O\left(\sqrt{\frac{N}{M}}\right).$$

Problem 4:

(a) Take $N = 16$ and let f mark exactly two inputs, for example 0000 and 0001. Then $M = 2$ and $k = \sqrt{N} = 4$ gives

$$\Pr(\text{success}) = \sin^2\left(9 \arcsin \sqrt{\frac{2}{16}}\right) = \frac{25}{2048} \approx 0.0122.$$

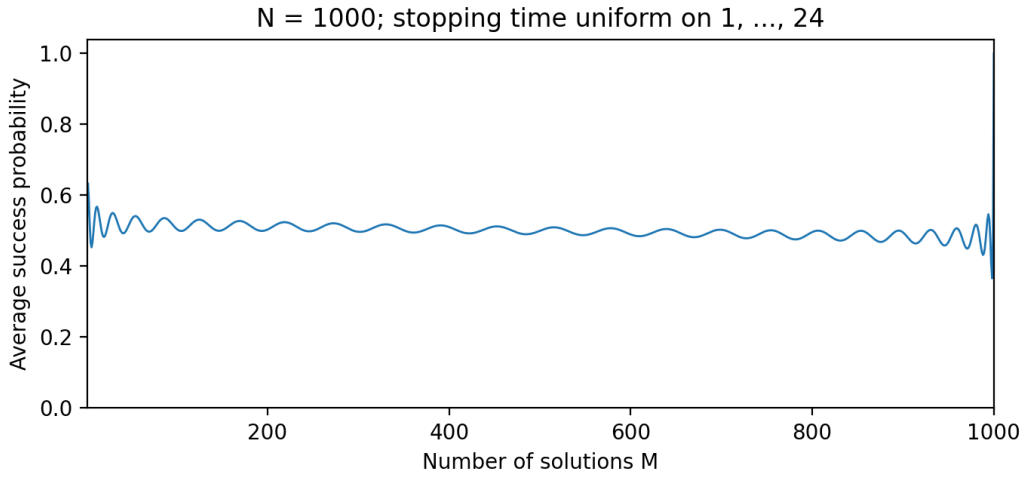
Only about 1.22% of runs succeed.

(b) Since the stopping time is an integer, set $L = \lfloor \pi\sqrt{N}/4 \rfloor$ and choose k uniformly from $1, \ldots, L$. The normalized average is

$$p(M) = \frac{1}{L} \sum_{k=1}^{L} \sin^2\left((2k+1) \arcsin \sqrt{M/N}\right).$$

For $N = 1000$, $L = 24$. The average is near $1/2$ for most M, rather than being close to zero for some badly chosen fixed stopping time. Numerically, its minimum is about 0.3660, at $M = 998$. At $M = 1000$, every input is marked, so success is 1.

The worksheet's expression suppresses the rounding of the upper limit. Once it is rounded down to L, use $1/L$ for a uniform average.



(c) Assume at least one solution exists. Run the randomized-stopping algorithm independently, checking each measured candidate x by evaluating $f(x)$. Stop only when the check succeeds.

To cover the case of many solutions as well, first try one uniformly random input in each round. If $M \geq N/4$, this alone succeeds with probability at least $1/4$. Otherwise, randomized Grover stopping has a constant success probability. Small constant-size domains can be checked directly.

If a round succeeds with probability at least a constant $p_0 > 0$, then r independent rounds fail with probability at most $(1 - p_0)^r$. A constant number of rounds gives any fixed target success probability, such as 99%, using $O(\sqrt{N})$ queries. For failure probability at most δ, use $O(\sqrt{N} \log(1/\delta))$ queries. If $M = 0$, no algorithm can output a marked input.

Problem 4(b): Python code

This code computes the average over all allowed integer stopping times; it does not estimate the average by sampling.

```
from pathlib import Path
import numpy as np
import matplotlib.pyplot as plt

N = 1000
L = max(1, int(np.floor(np.pi * np.sqrt(N) / 4)))
M = np.arange(1, N + 1)
theta = np.arcsin(np.sqrt(M / N))
k = np.arange(1, L + 1)

angles = theta[:, None] * (2 * k[None, :] + 1)
p = np.mean(np.sin(angles) ** 2, axis=1)

fig, ax = plt.subplots(figsize=(7, 3.4))
ax.plot(M, p, linewidth=1)
ax.set(
    xlabel="Number of solutions M",
    ylabel="Average success probability",
    title=f"N = {N}; stopping time uniform on 1, ..., {L}",
    xlim=(1, N),
    ylim=(0, 1.04),
)
fig.tight_layout()
fig.savefig(Path("grover_random_stopping.png"), dpi=200)
plt.show()

j = int(np.argmin(p))
print(f"L = {L}")
print(f"Minimum = {p[j]:.8f}, at M = {M[j]}")
print(f"At M = N: {p[-1]:.8f}")
```

Output:

```
L = 24
Minimum = 0.36597862, at M = 998
At M = N: 1.00000000
```