CSE622/622A IQC: Homework 4

Instructor: Debajyoti Bera (IIIT-Delhi)

Due: 31 March 2026

Q1 [7 points] You are encouraged to use Python sympy module for the algebraic operations. Submit a PDF containing the code and the results.

(a) (1 point) Read about Grover's algorithm, in particular, how the Grover iterator is implemented using the diffusion operator and the phase version of the oracle and the diffusion operator. Draw the 2-qubit quantum circuit for searching in a 4 element array; the circuit should use single-qubit gates and CNOT gate.

(b) (1 point) Suppose you are searching in a 4-element 0-1 valued array $X[0 \dots 3]$. You can write the phase-oracle $\hat{U}$ as an operator that maps

$$|i\rangle \mapsto (-1)^{X_i} |i\rangle = (1 - 2X_i) |X_i\rangle .$$

Express the operator U as a sum of outer-products.

(c) (1 point) Write down the 4×4 matrix corresponding to the Grover iterator.

(d) (1 point) Write down the state of the system after applying the Grover iterator once.

(e) (1 point) Write down the state of the system after applying the Grover iterator twice.

(f) (2 points) After applying Grover iterator twice, measure the state in the standard basis. Explain the measurement outcome(s) for $X = [0, 0, 0, 0]$ and $X = [1, 0, 0, 0]$.

For the next problem, you may use the following unordered search algorithms as subroutines. Here, N denotes the size of the array and m denotes the number of good solutions it has; m need not be known.

Grover If m is known, the **Grover** algorithm can solve the unordered search problem with probability of error less than m/N using $O(\sqrt{N/m})$ queries to A. If $m = 0$, it retuns "none" with no error.

ExactGrover If m is known, then the **ExactGrover** algorithm can solve the unordered search problem with no error using $O(\sqrt{N/m})$ queries to A.

GroverLV The **GroverLV** algorithm can solve the unordered search problem using expected $O(\sqrt{N/m})$ calls to A. It does not require knowledge of m, but if $m = 0$, the algorithm runs forever.

GroverMC The **GroverMC** algorithm can solve the unordered search problem using $O(\sqrt{N/m})$ calls to A. If $m > 0$, it returns a good solution with probability at least $2/3$, and if $m = 0$, it always returns "none".

Q2 [8 points] This is a question on unordered search. You are given query access to a binary array A and you have to find the index of any "1" in the array. If A has no "1", it should return "none".

You will first design two separate algorithms for two different cases.

(a) **(3 points)** Suppose you are told a number s such that A has at most s solutions. Design a quantum algorithm that finds a good solution with probability 1, using $O(\sqrt{sN})$ queries to A.

(b) **(3 points)** Now, suppose you are told a number s such that A has at least $s+1$ solutions. Design a quantum algorithm that finds a good solution with probability at least $1-2^{-s}$, using $O(\sqrt{sN})$ queries to A.

(c) **(2 points)** Finally, you will design an algorithm for the original problem by combining the two earlier algorithms. Let $\epsilon = 1/2^r$ for some $r \in (1, N)$ representing the desired probability of error. Design a quantum algorithm that makes $O(\sqrt{N \log(1/\epsilon)})$ queries to A [1] and solves the search problem with probability at least $1-\epsilon$, i.e., if there is no solution, it should always return "none", and if A has "1", it may still return "none" with error at most ϵ.
Hint: Guess s.

For all the algorithms questions above, write the algorithm, analyse the number of queries, and analyse the probability of error.

Q3 [10 points] The task of this question is to design a Qiskit-based quantum solver for Knapsack and use them to solve the instances on `https://www.kaggle.com/datasets/warcoder/knapsack-problem/data`.

(a) **[0 point], DO NOT SUBMIT** Learn about the Knapsack problem and how to convert equality and inequality constraints using slack variables [2]

(b) **[1 point]** Write a BruteForce(value array, weight array, capacity) function that takes a 5-item knapsack instance and returns the optimal value by doing a brute-force search over all possible sets of items.

(c) **[1 point]** Explain how to write a QUBO (minimization form) and the corresponding Ising Hamiltonian for the following Knapsack instance: values [1 41 47 7 26], weights [1 6 15 6 16], capacity 86.

For the rest of the questions, assume that the Knapsack instances are for 5 items only.

(d) **[2 points]** Implement a QUBO_to_Operator(QUBO) function that takes a QUBO (in any suitable format) and returns a Hamiltonian (as a list of coefficients of Pauli operators [3]). Before returning, print the Hamiltonian in a human-readable format (ex: "$-5 + 2X_1Y_3 - \ldots$").

(e) **[4 points]** Implement a Solve(Hamiltonian) function that uses VQE [4] to return the ground state of the Hamiltonian [5].

(f) **[2 points]** Use Solve on each of the Knapsack instances on the Kaggle website. Finally, plot a table showing the following for each instance:

(a) instance (value array, weight array, threshold)

(b) optimum value (using BruteForce)

(c) value obtained using Solve

(d) accuracy (obtained / optimum as percentage)

[1] Most efficient classical approaches have query complexities of the form $O(\ldots \log(1/\epsilon))$.

[2] References:
`https://quantumcomputing.stackexchange.com/questions/24010/penalty-formulation-regarding-inequalitties`, `https://docs.dwavequantum.com/en/latest/quantum_research/reformulating.html` (constraints Sections)

[3] A 2-qubit Hamiltonian, e.g., can be always written as $\sum_{i,j=0}^{4} c_{ij} P_i \otimes P_j$, where $P_i, P_j \in \{I, X, Y, Z\}$ using constants $[c_{ij}]$

[4] VQE API: `https://qiskit-community.github.io/qiskit-algorithms/stubs/qiskit_algorithms.VQE.html` requires an ansatz (for which you can use any of the readily available ansatz in Qiskit), an estimator to calculate "loss" for which you can use Qiskit EstimatorV2, and an optimizer for which you can use optimizer available in Qiskit.

[5] You can follow the steps outlined in
`https://quantum.cloud.ibm.com/learning/en/courses/quantum-diagonalization-algorithms/vqe`.