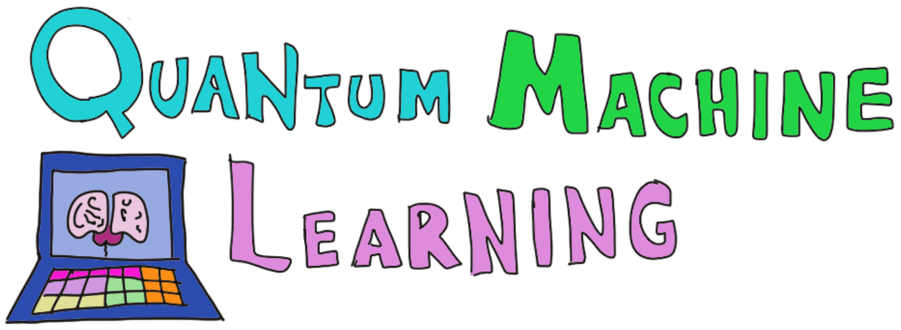

QML: The Unit Disk Maximum Independent Set (UDMIS) Problem [500 points]

Version: 1

Quantum Machine Learning

In this set of challenges, you'll explore methods and applications for training variational quantum circuits and quantum neural networks. Both play critical roles in quantum machine learning. They typically have a layered structure, and a set of tunable parameters that are learned through training with a classical optimization algorithm.

The circuit structure, optimization method, and how data is embedded in the circuit varies heavily across algorithms, and there are often problem-specific considerations that affect how they are trained. In the **Quantum Machine Learning** challenges, you'll explore how to construct and optimize a diverse set of circuits from the ground up, and become acquainted with some of today's popular quantum machine learning and optimization algorithms.

Problem statement [500 points]

Although finding ground states of Hamiltonians is the bane of a physicist's existence, some Hamiltonians' ground states map directly to real-world problems; approximation methods to finding ground states don't just serve scientific discovery. Take, for instance, this Hamiltonian,

$$\hat{H} = -\sum_{i \in V} \hat{n}_i + u \sum_{(i,j) \in E} \hat{n}_i \hat{n}_j,$$

where $G = (V, E)$ denotes a graph with vertices V and edges E and the operators $\hat{n}_i$, often called "occupation" operators, linearly map to the familiar $\hat{\sigma}^z$ operator,

$$\hat{n}_i = \frac{\hat{\sigma}_i^z + 1}{2}.$$

The single-body term in the Hamiltonian favours all vertices to be occupied, while the two-body term penalizes connected vertices from both being occupied. How we deem which vertices are connected depends on some user-defined constraints. For instance, consider the "unit disk" constraint, where vertices within one unit of distance between each other are said to be "connected" by an edge E. Finding the ground state of such a Hamiltonian with those constraints is equivalent to solving a well-known problem called the Unit Disk Maximum Independent Set (UDMIS) problem. Formally, the UDMIS problem is as follows.

Let $G = (V, E)$ be a graph defined by vertices V and edges E. Bit strings are given by $S = (n_1, \cdots, n_N)$, where $n_i \in 0, 1$, $N = |V|$, and whose Hamming weight is $|S| = \sum_{i=1}^{N} n_i$. The UDMIS problem solved by finding $|S^*| = \max_{S \in B} |S|$ such that S is an *independent set* and is subject to the unit-disk constraints, where B is the set of all possible bit strings. An *independent set* is defined as a set whose vertices connected by an edge are not both occupied: with mutually non-connected vertices: $(n_i, n_j) \neq (1, 1)$ if $(i, j) \in E$. An example is given in Figure 1.

You are challenged with programming this Hamiltonian in PennyLane and variationally optimizing a circuit that finds the ground state of the Hamiltonian given a unit-disk graph G for $|V| = 6$ vertices. Specifically, your code will:

- Construct the Hamiltonian in such a way that PennyLane can interpret it as an observable.
- Define a variational circuit.
- Optimize the variational circuit with respect to the expectation value $\langle \hat{H} \rangle$.

The provided template file `udmis_template.py` contains a few functions:

- `hamiltonian_coeffs_and_obs`: This is where you will need to provide PennyLane instructions for how to create the Hamiltonian in question. See this for helpful hints.
- `edges`: This computes the edges of the given graph subject to the unit-disk constraints. It returns the number of edges and a symmetric boolean matrix that represents the existence of an edge between indices (i.e. `E[2,3] = True` means there is an edge between vertices 2 and 3).
- `variational_circuit`: Where you will need to create a variational circuit that will be optimized.
- `train_circuit`: Where the variational circuit is trained by minimizing the `cost` function. The output of this function is the energy density (i.e. energy divided by $|V| = 6$).

The value of u in the above Hamiltonian is hard-coded to 1.35 since it works for the purposes of this problem.

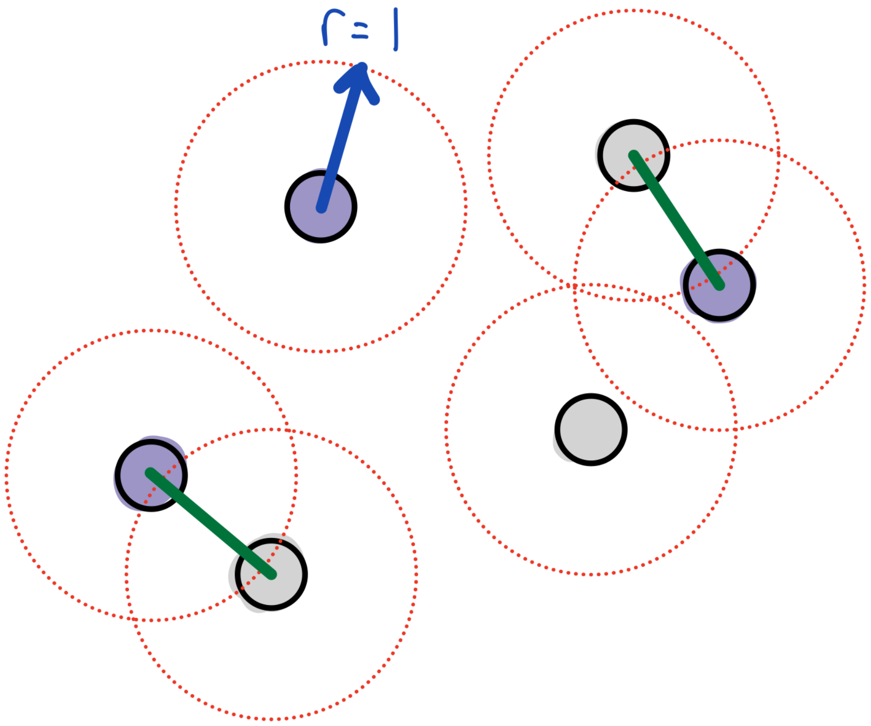

Figure 1: An example solution to the UDMIS problem for a given graph. Purple/grey vertices are occupied/unoccupied.

Input

- `list(float)`: A list of coordinates that define the graph G.

Output

- `float`: The energy density that your optimization routine computes.

Acceptance Criteria

In order for your submission to be judged as "correct":

- The outputs generated by your solution when run with a given `.in` file must match those in the corresponding `.ans` file to within the `0.001` tolerance specified below. To clarify, your answer must satisfy

$$\text{tolerance} \geq \left| \frac{\text{your solution} - \text{correct answer}}{\text{correct answer}} \right|.$$

- Your solution must take no longer than the `Time limit` specified below to produce its outputs.

You can test your solution by passing the `#.in` input data to your program as stdin and comparing the output to the corresponding `#.ans` file:

```
python3 {name of file}.py < 1.in
```

WARNING: Don't modify the code outside of the `# QHACK #` markers in the template file, as this code is needed to test your solution. Do not add any print statements to your solution, as this will cause your submission to fail.

Specs
Tolerance: **0.001**
Time limit: **80 s**

Version History

Version 1: Initial document.