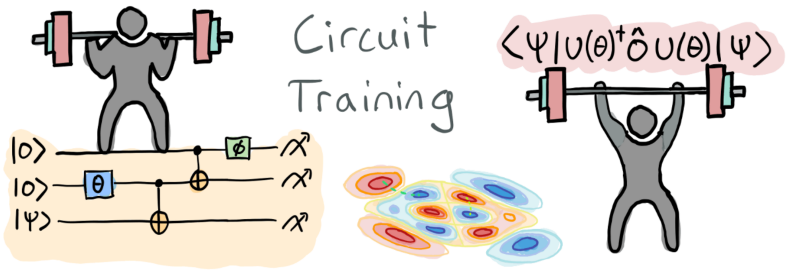

Circuit Training: QAOA [200 points]

Version: 1

NOTE: Coding templates are provided for all challenge problems at this link. You are strongly encouraged to base your submission off the provided templates.

Overview: Circuit Training challenges

In this set of challenges, you'll explore methods and applications for training variational quantum circuits. Variational circuits play a role in quantum machine learning akin to that of neural networks in classical machine learning. They typically have a layered structure, and a set of tunable parameters that are learned through training with a classical optimization algorithm. Data is input to the circuit by embedding it into the state of a quantum device, and measurement results inform the classical optimizer how to adjust the parameters.

The circuit structure, optimization method, and how data is encoded in the circuit varies heavily across algorithms, and there are often problem-specific considerations that affect how they are trained. In this set of 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 [200 points]

This challenge focuses on the quantum approximate optimization algorithm (QAOA) and explores how it can be used to solve a graph-based problem. If you are unfamiliar with QAOA, check out our tutorial for more information!

Your objective is to set up a QAOA circuit in PennyLane and use pre-optimized parameters to identify the maximum independent set of a graph with six nodes.

To do this, the first step should be to fix the cost and mixer Hamiltonians for the maximum independent set problem. This can be done using PennyLane's qaoa module. Note that you should use the *constrained* version of the maximum independent set Hamiltonian.

You should then set up the QAOA circuit. The circuit should be composed of alternating cost and mixer layers with an overall depth of ten, as shown in the diagram below. The input quantum state to the QAOA layers should be the $|000000\rangle$ state, i.e., so that Hadamard gates are *not* placed before the layers. The cost and mixer layers should be controlled by $\boldsymbol{\gamma}$ and $\boldsymbol{\alpha}$ parameters, each of length ten. You will be provided with *optimized* parameters and will not need to optimize further.

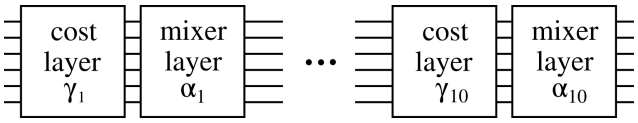

Figure 1: Ten QAOA layers

The output of the circuit should be the measurement outcome probabilities of the computational basis states. The maximum independent set can be found by identifying the most probable basis state and finding the corresponding subset of nodes from the graph. For example, suppose that the most likely computational basis state turns out to be $|011010\rangle$. This will tell us that we should pick out nodes 1, 2 and 4 from our graph as a solution.

Input

You will be provided with an input graph and parameters $\boldsymbol{\gamma}$ and $\boldsymbol{\alpha}$. These parameters have been pre-optimized for the QAOA layers shown in the diagram above when using the cost and mixer Hamiltonian from the constrained maximum independent set problem.

You should work on the `find_max_independent_set` function. This function has the input graph and optimized parameters as inputs. The graph will be specified as a NetworkX object and the parameters as a single NumPy array of shape `(2, 10)`, with the first row corresponding to $\boldsymbol{\gamma}$ and the second to $\boldsymbol{\alpha}$. You can optionally plot the graph to help as you work on the challenge, but make sure to turn off this feature when you submit.

Output

The output of the `find_max_independent_set` function should be the maximum independent set, returned as a list of unique integers in increasing order, specifying the subset of nodes in the graph. Note that the first node of the graph is labelled as node 0. A unique solution exists for each of the provided graphs.

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 `Tolerance` specified below.
- 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 circuit_training_200_template.py < 1.in
```

WARNING: Don't modify any of the code in the template outside of the `# QHACK #` markers, 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: **Exact solution required** Time limit: **30 s**

Version History

Version 1: Initial document.