WON0004 - Introduction to Quantum Computing (Cohort) – Spring 2023

QC Seminar 1 – Solving MaxCut with QAOA

Assignments:

1. **Network graphs:**
   i) In the simulator notebook notebook3_simulators.ipynb create different 4-, 5-, 6-node network graphs with different (non-trivial, non-symmetric) weight assignments and solve the maxcut problem with QAOA using the three open-source simulators (Cirq, Qulacs, Symbolic) and the QiskitWavefunctionSimulator().

```
def create_graph():
    graph = nx.Graph()

    graph.add_nodes_from([0, 1, 2, 3, 4])
    ...

    return graph
```

   ii) Use a quantum circuit with 2 layers:

```
qaoa = QAOA.default(cost_hamiltonian=hamiltonian, n_layers=2)
```

   iii) Perform measurements with 10000 samples:

```
measurements = runner.run_and_measure(circuit, n_samples=10000)
```

   iv) Use the 'in_process' option: **workflow_run = run_max_cut().run("in_process")**
   v) Run each simulator 5 times. Save the PNG images of the different solution graphs for each simulator and add a description where you list the name of the simulator, the execution time, list the nodes in each subset of the solution. Add the initial 4-, 5-, 6-node network graph you created. *Tip: You can create the initial network with Python's networkx library. To get ideas see e.g. the link on network graphs given in the instructions file.*

2. **Sample numbers:**
   i) Select again the original 6-node network graph used in the simulators notebook.
   ii) Use **QulacsSimulator().** Increase the sample number in steps **n_samples = 1000, 10000, 100000, 10^6.** Record the execution time. Record the list of nodes in each subset of each solution. Run each setting at least 5 times.
   iii) Do you see one particular solution appear more often when you increase the sample size? If not, what do you think could be the reason for that?

3. **Circuit depth:**
   i) Use the same settings as under problem number 2. using the original 6-node network graph in the simulators notebook with the **QulacsSimulator()** with10000 samples.
   ii) Run the quantum circuit for 1, 2 and 3 layers at least 5x in each case. Record again the execution times and returned solutions. Example:

```
qaoa = QAOA.default(cost_hamiltonian=hamiltonian, n_layers=2)
```

   iii) Do you see a higher probability for one particular solution when you increase the number of circuit layers?
   iv) Re-do the experiments above with 100,000 samples. Does the situation change?

4. **Quantum backend:**

i) Motivation: We now want to run a simulation or two using an actual IBM quantum backend. It can take a very long time until a job submitted to the IBM queue gets executed. So this exercise might take some time to complete.

ii) Setup a smaller 5-node network graph. This is so you can submit the maxcut-QAOA workflow to a number of IBM quantum backends available to with 5 or 7 qubits available.

```
@sdk.task
def create_graph():
    graph = nx.Graph()
    graph.add_nodes_from([0, 1, 2, 3, 4, 5])
    graph.add_edge(0, 1, weight=10)
    graph.add_edge(0, 3, weight=5)
    graph.add_edge(1, 2, weight=1)
    graph.add_edge(2, 3, weight=1)

    graph.add_edge(0, 4, weight=10)
    graph.add_edge(0, 5, weight=5)
    graph.add_edge(1, 4, weight=1)
    graph.add_edge(2, 4, weight=5)
    graph.add_edge(1, 5, weight=5)
    return graph
```

iii) For comparison purposes run the quantum circuit first with one of your local simulators e.g **CirqSimulator()** a few times and record the solutions. Select sample size = 10. Use the alternative QAOA solution method that is later also necessary for a run on the quantum backend.

```
runner = CirqSimulator()
qaoa = QAOA.default(cost_hamiltonian=hamiltonian, n_layers=2,
                    use_exact_expectation_values=True)
```

iv) Now switch to the setup for a quantum backend. From the available resources pick the 5-quubit quantum hardware **'ibmq_lima'** and one of the 7-quubit options 'ibm_olso' or **'ibm_nairobi'.**

```
runner = create_ibmq_runner(sdk.secrets.get("IBMTOKEN", config_name="eval-12"), "ibmq_lima")
# runner = create_ibmq_runner(sdk.secrets.get("IBMTOKEN", config_name="eval-12"), "ibm_nairobi")
# runner = create_ibmq_runner(sdk.secrets.get("IBMTOKEN", config_name="eval-12"), "ibm_oslo")
```

In the QAOA turn off the expectation values setting in contrast to the simulator run above (on quantum backends solutions have to be evaluated via actually re-taking shots with the hardware and prepared physical qubits):

```
qaoa = QAOA.default(cost_hamiltonian=hamiltonian, n_layers=2,
                    use_exact_expectation_values=False, n_shots=10)
```

Try to complete at least one run for each hardware. Again, this might take a very long time to complete due to job queuing. You might have to submit and re-check the submission in your IBM account under jobs submission dashboard at https://quantum-computing.ibm.com/jobs.

v) For comparison purposes also submit a workflow using one of the online available IBM simulators, **'simulator_statevector'**.

vi) In each case choose **n_layers=2, n_shots=10, n_samples=10** as parameters. Record the solutions you obtain with the execution times.