Lab 3 Qubit Allocation and Routing Algorithms

CS7400 Online
Georgia Institute of Technology
(Dated: Summer 2025)

For each question, you need to complete the associated `question*.py` file by implementing the required functions for that question. Submit `question1.py`, `question2.py`, and `question3.py` to Gradescope Lab-3.

I. DECOMPOSITION OF BENCHMARK CIRCUITS

In the Lab-3/SWAP-Benchmarks, we have four quantum benchmark circuits in ".qasm" format. Unfortunately, not all of them use the basis gate set that IBM uses for physical execution. This problem can be solved by picking a basis set of gates, and every gate in the benchmark circuit can be expressed as a composition of gates in the basis set, this process is called the decomposing the circuit.

A. Basis gates set [cx,u]

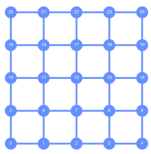
In `q1a` function in `question1.py`, decompose the benchmark circuits using qiskit transpile function with basis gates set [cx,u], and return a dictionary of benchmark circuits' names as keys and transpiled circuits as values. You are expected to use the same names of circuits generated by `get_benchmark_dict`.

B. Basis gates set [cx,rx,ry,rz]

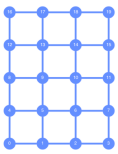
In `q1b` function in `question1.py`, decompose the benchmark circuits using qiskit transpile function with basis gates set [cx,rx,ry,rz], and return a dictionary of benchmark circuits' names as keys and transpiled circuits as values. You are expected to use the same names of circuits generated by `get_benchmark_dict`.

II. SABRE ALGORITHM FOR DEVICE TOPOLOGY

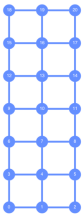
In this question, we learn how SABRE algorithm can be used to map quantum circuits to given quantum device topology. We will use qiskit transpile function to apply SABRE algorithm, and evaluate the change of circuit depth and the number of SWAPs inserted for each benchmark circuits averaged over all coupling maps:



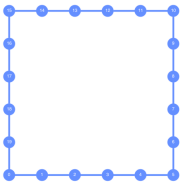
(a) Grid 5x5



(b) Grid 5x4



(c) Grid 7x3



(d) Ring 20

A. Create coupling maps

In `create_coupling_maps` function in `question2.py`, generate each coupling map as shown above using transpiler.CouplingMap and store them in the dictionary provided using the given keys.

B. Average depth change

In `average_depth_change` function in `question2.py`, calculate the average change of circuit depth when transpiling the benchmark circuits for given coupling maps, using SABRE for routing and mapping. You need to return a dictionary with benchmark circuits' names as keys and change of circuit depth averaged over all coupling maps. You are expected to use the same names of circuits generated by `get_benchmark_dict`.

C. Average number of swaps change

In `average_nswap_change` function in `question2.py`, calculate the average number of swaps inserted when transpiling the benchmark circuits for given coupling maps, using SABRE for routing and mapping. You need to return a dictionary with benchmark circuits' names as keys and number of swaps inserted averaged over all coupling maps. You are expected to use the same names of circuits generated by `get_benchmark_dict`.

III. SIMULATION OF A NOISY DEVICE

As shown in our lectures, real quantum machines suffer from noisy gates. From Lab-2 we learned that we can simulate noisy gates in qiskit using U gates. In this question, you need to first implement both noise-free and noisy CNOT gates, then practice qubit allocation by simulating a NISQ machine using your implementation of noisy CNOT gates.

For simplicity, we define a noisy CNOT gate to act in the following way: A noisy CNOT gate with fidelity p, is a 2-qubit quantum gate, when the control qubit is at state 0, applies identity operation on the target qubit, when the control qubit is at state 1, applies NOT operation on the target with probability p and applies identity operation on the target with probability $1 - p$.

A. Implementation of a noise-free CNOT gate

In `cx_ideal` function in `question3.py`, implement a noise-free CNOT gate without using `CXGate`. You must return a `qiskit.circuit.Gate` object.

B. Implementation of noisy CNOT gates

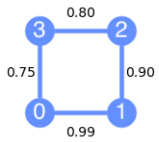
In `cx_95` function in `question3.py`, implement a noisy CNOT gate as defined above with fidelity $p = 0.95$, You must return a `qiskit.circuit.Gate` object. Your implementation must follow the definition described in this question.

In `cx_70` function in `question3.py`, implement a noisy CNOT gate as defined above with fidelity $p = 0.70$, You must return a `qiskit.circuit.Gate` object. Your implementation must follow the definition described in this question.

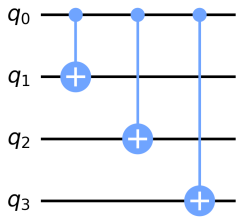
C. 4-qubit NISQ machine

The figure below demonstrates a 4-qubit NISQ machine coupling map similar to the one we have seen in our lecture, where the value of each edge denotes the fidelity of a CNOT gate (in both directions) as defined in this question. We would like to run the following logical quantum circuit on this 4-qubit NISQ machine.

In `q3c_edge_gates` function in `question3.py`, create noisy CNOT gates defined in this question based on the device topology. In `q3c_device_compatible_physical_circuit` function in `question3.py`, implement the mapped physical quantum circuit on the coupling map. You need to consider the optimal way to allocate the qubits to maximize the overall fidelity. You can only use 2-qubit noisy CNOT gates in `q3c_edge_gates`. In `q3c_device_qubit_route_mapping` function in `question3.py`, provide your logical qubit to physical qubit mapping in a dictionary in the following format `logical_qubit: physical_qubit`, this is the dictionary you use to allocate logical qubits in the circuit to the physical qubits in the device. In `q3c_device_qubit_read_mapping` function in `question3.py`, provide your physical qubit to logical qubit mapping in a dictionary in the following format `physical_qubit : logical_qubit`, this is the dictionary you use to read from the physical device post execution and retrive the result for the logical quantum circuit. If a physical qubit is not used, you should set its value to `None`.



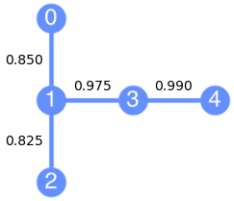
(a) 4-qubit NISQ machine



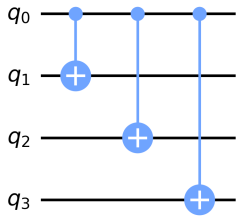
(b) Logical quantum circuit to run

D. 5-qubit NISQ machine

The figure below demonstrates a 5-qubit NISQ machine coupling map similar to the one we have seen in our lecture, where the value of each edge denotes the fidelity of a CNOT gate (in both directions) as defined in Q3. We would like to run the following logical quantum circuit on this 5-qubit NISQ machine.



(a) 5-qubit NISQ machine



(b) Logical quantum circuit to run

In `q3d_edge_gates` function in `question3.py`, create noisy CNOT gates defined in this question based on the device topology.

In `q3d_device_compatible_physical_circuit` function in `question3.py`, implement the mapped physical quantum circuit on the coupling map. You need to consider the optimal way to allocate the qubits to maximize the overall fidelity. You can only use 2-qubit noisy CNOT gates in 'q3d_edge_gates'.

In `q3d_device_qubit_route_mapping` function in `question3.py`, provide your logical qubit to physical qubit mapping in a dictionary in the following format `logical_qubit: physical_qubit`, this is the dictionary you use to allocate logical qubits in the circuit to the physical qubits in the device.

In `q3d_device_qubit_read_mapping` function in `question3.py`, provide your physical qubit to logical qubit mapping in a dictionary in the following format `physical_qubit : logical_qubit`, this is the dictionary you use to read from the physical device post execution and retrive the result for the logical quantum circuit. If a physical qubit is not used, you should set its value to 'None'.