Analysis on Grover's Algorithm on Qiskit Runtime Service and Enhancement with Error Mitigation

Rei Aoki
Iowa State University, raoki@iastate.edu

***Abstract* – This paper consists of three major sections: introduction to qiskit, overview of Grover's algorithm, and its analysis on the experiment. We first walk through what IBM Qiskit offers to a client what it can be executed and built on a real quantum hardware. It is essential to follow the workflow pattern to implement one's desired circuit and execute on a quantum hardware available on cloud service. On second, the overview of Grover's algorithm explains its procedures and necessary circuit components. On third, by implementing Grover's algorithm, one can run it on Qiskit and see the increased probabilities by giving increasing number of the amplitude amplification operations.**

INTRODUCTION TO QISKIT

Qiskit is an open-source collection of libraries containing useful tools to build and execute quantum circuits on quantum computers [1]. Among these software tools, part icularly essential ones are the open-source Qiskit SDK (Software Development Kit) and the runtime environment (called Qiskit Runtime) through which one can run programs on IBM® quantum processing units (or QPUs). The Qiskit SDK can be used for working on quantum hardware either statically, dynamically, or with scheduling at given quantum circuits, operators, and primitives. Once abstract circuits are built, one can transform and adapt them to suit a specific device topology. This process is technically called 'transpilation' by IBM, so we will use this term throughout the paper. It is noted that transpilation can optimize the specific circuit configurations for execution on QPUs. Also, one can utilize primitives that provide the modules, such as Sampler and Estimator, to calculate statistical data (sample data and expectation values, respectively). These interfaces offer benefits to different quantum hardware providers as their implementations differ.

For the runtime environment, Qiskit Runtime is a cloud-based service for running programs or quantum computations on IBM Quantum® hardware. The service is included in `qiskit-ibm-runtime` package for client usage. The Qiskit Runtime service also provides optimal implementations of the Qiskit primitives for quantum computations, making the workflow pattern smooth. Most notably, Qiskit Runtime is designed to employ additional resources, both classical and quantum, and suppression and error mitigation techniques, to output a higher-success-probability result from running quantum circuits and computations on QPUs. Several examples are dynamical decoupling, Pauli twirling, and twirled readout error extinction (TREX) and zero-noise extrapolation (ZNE), etc., for error suppression and mitigation [1]. The real quantum hardware is subject to errors based on specific qubits and connectivity on a device (Figure 1). It is important to exercise one of those techniques to obtain higher probability.

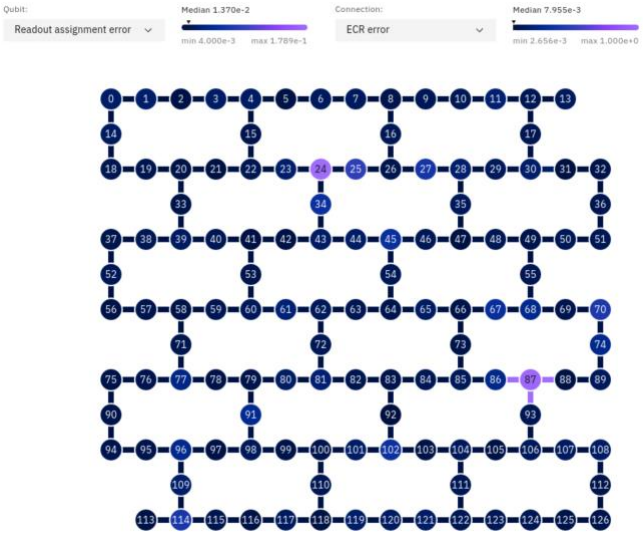

FIGURE 1
CALIBRATION DATA FOR IBM_BRISBANE

Qiskit Runtime also provides three kinds of execution modes for running quantum programs on the quantum hardware: *Job*, *Session*, and *Batch*, which have different implications for the quantum job queue and scheduling. A *Job* is a query to a primitive that can be executed over a certain number of shots [1]. *Sessions* allow one to efficiently run multiple jobs in iterative workloads on quantum computers [1]. *Batch* mode allows one to submit all one's jobs at once for parallel processing [1].

One particular open-source package is worth mentioning, even though it is not included in Qiskit, since it interfaces with Qiskit and provides supplementary

functionality to augment the Qiskit workflow pattern. Qiskit Aer (qiskit-aer) is a package for quantum computing simulators with realistic noise models that can run on a local classical computer without the need to connect to the runtime server. This simulation can be quite useful and helpful before running one's quantum computations on real hardware, so that one can check the correctness of the implementations.

OVERVIEW OF GROVER'S ALGORITHM

Before implementing Grover's algorithm, it might be helpful to describe what the algorithm can solve more efficiently than any classical computations. We compare the computational complexity of a simple classical algorithm with that of Grover's algorithm with a famous example problem. *Grover's algorithm* is a quantum search algorithm for unstructured search problems that offers a quadratic improvement over any related classical algorithms [2]. Suppose, for example, you are given a map containing many cities and wish to find the shortest route passing through all cities on the map. If there are *N* input routes, it takes *O(N)* operation times, by simply searching all possible routes, to return the output on a classical computer [3]. On the other hand, Grover's algorithm speeds up this search computation substantially, requiring only $O(\sqrt{N})$ time [3]. While the wide range of Grover's algorithm application seems imminent, it should be noted that the theoretical quadratic advantage it offers is not likely to be implemented in practice any time soon. The reason is that we have a classical computer that is technologically more advanced and available than currently available quantum computing devices. The quadratic improvement over classical algorithms offered by Grover's algorithm cannot yet overcome the menacing clock speeds of advanced classical computers for any unstructured search problem as of today.

GROVER'S ALGORITHM IMPLEMENTATION ON QISKIT

Since we have gained background knowledge about both Qiskit and our to-be-run algorithm, we start implementing Grover's algorithm codes and circuits. Let us first describe the steps the algorithm needs to take.

Grover's algorithm [2]

1. Initialize an *n* qubit register *Q* to the all-zero state $|0n\rangle$ and then apply a Hadamard operation to each qubit of *Q* [2].
2. Apply *t* times the unitary operation $G = H^{\otimes n} Z_{OR} H^{\otimes n} Z_f$ to the register *Q* [2]
3. Measure the qubits of *Q* with respect to standard basis measurements and output the resulting string [2].

Phase query gates Z_f is defined for the function *f* as [2]:

$$Z_f|x\rangle = (-1)^{f(x)}|x\rangle$$

Z_{OR} operator is defined as [2]:

$$Z_{OR}|x\rangle = \begin{cases} |x\rangle & x = 0^n \\ -|x\rangle & x \neq 0^n \end{cases}$$

Now, let us implement the Grover's circuit in Qiskit. A sample code is available in the reference [4]. We will follow the workflow pattern briefly outlined in the introduction. First, we need to call our backend and then use it to get all the configurations to inform a transpiler how to convert the abstract circuit that we build into a specific instruction-set-architecture-based circuits that can be executed on the device. The backend name will be the 'ibm_sherbrooke' with a 127 qubits device [5]. Then we pass all the parameters into a so-called pass manager with some optimization level, and it will handle transpiled circuits running on the backend. This will then populate information about the backend, such as the connectivity in the native gates (Figure 2).

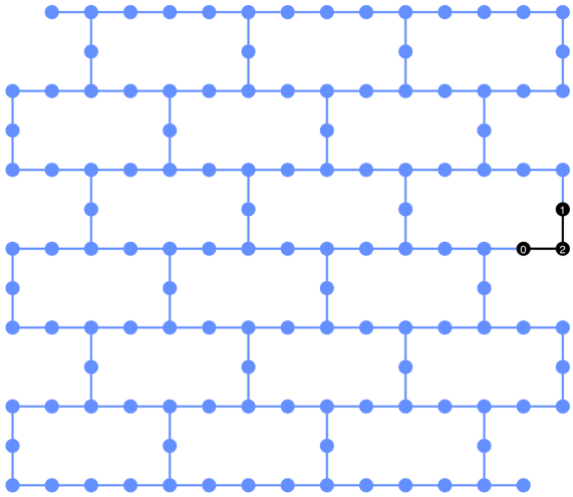

FIGURE 2
BACKEND (IBM_SHERBROOKE) CONNECTIVITY WITH 3 QUBITS USAGE

Once we set up the backend, we can build our circuits that correspond to Grover's oracle, which presets one or more marked computational bases, with its phase being -1. This, in fact, becomes the solution we want to solve. We choose our marketed state to be the binary representation of 5, or 101. The coding is in the following figure (Figure 3):

```
marked_states = ["101"]

oracle = grover_oracle(marked_states)
oracle.draw(output="mpl", style="iqp")
```

FIGURE 3
MARKED STATE SET TO '101'

If we take a look at the generated Grover's oracle, the markted basis state is effectively applied with two X gates before and after the controlled-Z gate (Figure 4).

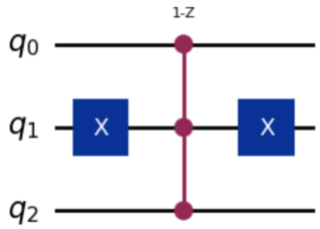

FIGURE 4
GROVER'S ORACLE '101'

The essential part of Grover's algorithm is the use of the amplitude amplification on the oracle. The greater the number of times we apply the amplitude amplification to the overall circuit, the higher the success probability we obtain for the marked state. In our case, this can be achieved by applying additional Hadamard, X, and CNOT-X gates (Figure 5).

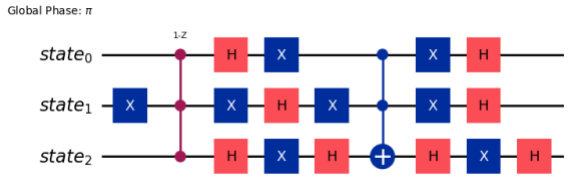

FIGURE 5
AMPLITUDE AMPLIFICATION OF GROVER'S ORACLE

There is an optimal number of iterations that we can apply the amplitude amplification to the circuit which is defined as

$$t_{optimal} = \left\lfloor \frac{\pi}{4sin\sqrt{n} \cdot 2n} \right\rfloor$$

Finally, the complete Grover's circuit is realized by applying Hadamard gates to every input qubit at the beginning, creating superposition states with equal probability for each qubit (Figure 6). In our experiments, we set the number of qubits, $n = 3$, and the optimal iteration number,
$t = 2$ (we measure from 0 to 2). Also, we are required to measure the output in the end, so we place observables, Pauli ZZ strings to each qubit and the measuring qubit (in our case, the fourth qubit). When mapping our observables to the qubits in the IBM backend device, the ordering will get swapped around, so we need to ensure each index in the right position on the transpiled circuits. This can be done by calling

operator.apply_layout(transpiled_circuit.layout)

for all observable operators.

Since our circuit implementation is complete, we analyze its behavior with different numbers of iterations. In the next analysis section, we show how the number of different iterations we apply to the Gover's oracle, we obtain a bit-string with the most probable result.

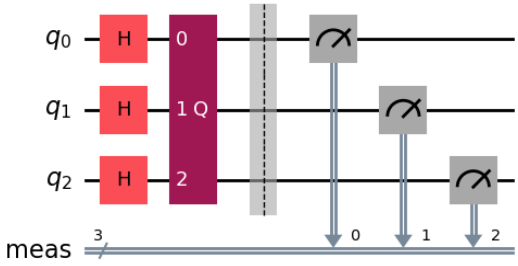

FIGURE 6
COMPLETE GROVER'S CIRCUIT WITH N = 3 AND OBSERVABLES

ANALYSIS ON GROVER'S ALGORITHM

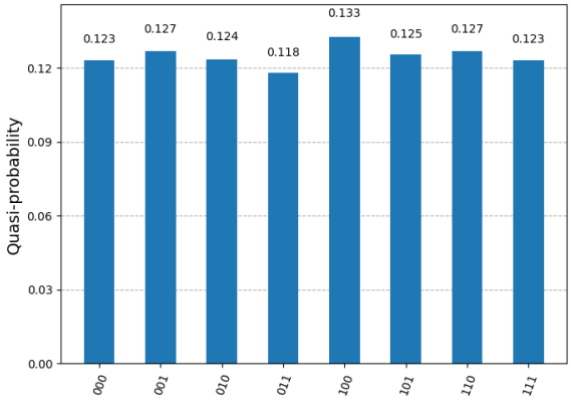

FIGURE 7
T = 0: SUPERPOSITION STATES
WITH NEAR-EQUAL PROBABILITY

Starting from $t = 0$, the circuit is applied without any operations from the Grover's oracle or the amplitude amplification (Figure 7). The resulting output creates superposition states for 3 qubits, each having nearly equal probability of around 0.125. This observed data agrees with the expectation since we only apply Hadamard gates on all input qubits.

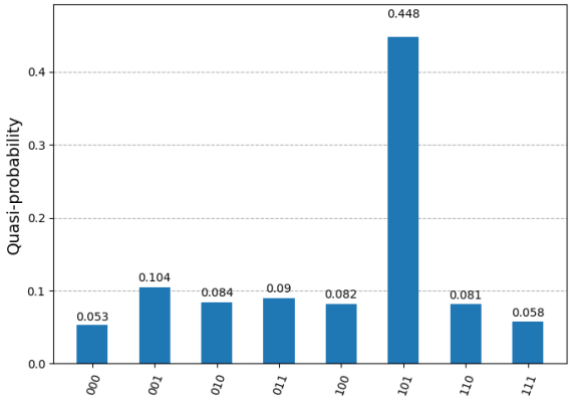

FIGURE 8
T = 1: ONE ITERATION OF AMPLITUDE AMPLIFICATION

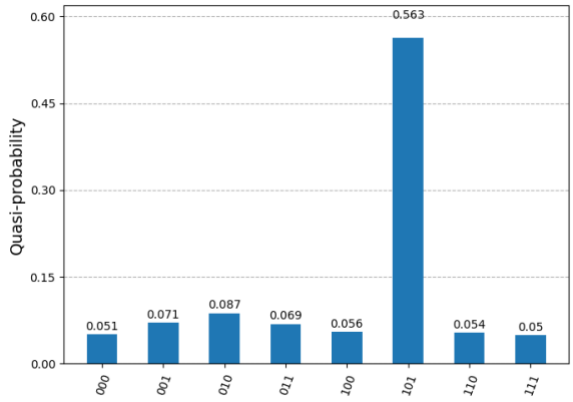

FIGURE 9
T = 2: TWO ITERATION OF AMPLITUDE AMPLIFICATION

When $t = 1$, the resulting output favors the marked state, 101, with the probability of 0.448 (Figure 8). When $t = 2$, the circuit outputs the sample data with the probability of 0.563 for the marked state (Figure 9). The expectation of increasing probability for the marked state is observed correctly when we increase the number of iterations to apply the amplification operations. However, it is noted that the probability for each t is relatively low compared to the theoretical calculation. A local simulation outputs the probabilities of 0.741 and 0.933 for $t = 1$ and 2, respectively. The significant differences in data are due to the presence of errors in the quantum device.

In the last section, we will carry out the error mitigation technique called dynamical decoupling to see if the result can be refined further to produce a higher probability of the marked state.

AFTER DYNAMICAL DECOUPLING

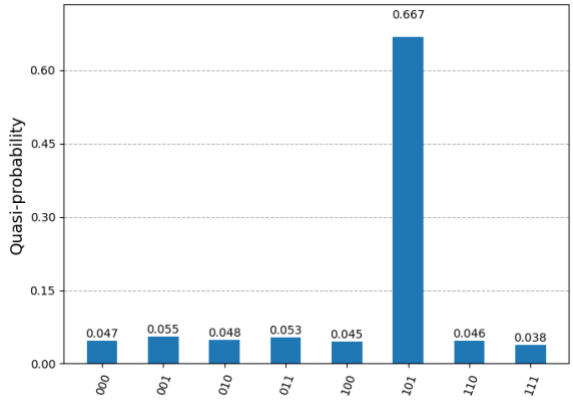

FIGURE 10
T = 2: THE EFFECT OF DYNAMICAL DECOUPLING

Dynamical decoupling extends the life of qubit coherence by inserting additional pulse sequences on idling qubits to nearly clear out the effect of these errors [6]. Each inserted pulse sequence can be treated as an identity operation, but the physical presence of the pulses suppresses errors [6]. Now the error-corrected data shows the slightly higher probability of 0.667 on the marked state.

Furthermore, if we increase the number of input qubits to n = 4, we see that the probability is scattered over all qubits. Citing from *Indian Journal of Pure & Applied Physics*, "as the number of qubits increases from 2 to 4-qubit, the circuit complexity increases which in turn increase the measurement errors at the time of measurement" [7]. Hence, we leave $n = 3$ and $t = 2$ to be our best experimental inputs and result.

REFERENCES

[1] Intro to Qiskit https://docs.quantum.ibm.com/guides
[2] Grover's Algorithm https://learning.quantum.ibm.com/course/fundamentals-of-quantum-algorithms/grovers-algorithm
[3] Nielsen & Chuang
[4] Grover's Algorithm Tutorials https://learning.quantum.ibm.com/tutorial/grovers-algorithm

[5] IBM Quantum Platform https://quantum.ibm.com/
[6] Error mitigation and suppression techniques https://docs.quantum.ibm.com/guides/error-mitigation-and-suppression-techniques
[7] *Indian Journal of Pure & Applied Physics* https://inspirehep.net/files/706be950135d5648808169ec2632298f

AUTHOR INFORMATION

Rei Aoki, Junior student, Department of Computer Science, Iowa State University.