NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

EVALUATION OF A PHYSICAL QUANTUM COMPUTER

by

John A. Heropoulos

June 2020

Thesis Advisor:	Theodore D. Huffmire
Second Reader:	Francesco A. Narducci

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** June 2020	**3. REPORT TYPE AND DATES COVERED** Master's thesis

4. TITLE AND SUBTITLE EVALUATION OF A PHYSICAL QUANTUM COMPUTER	**5. FUNDING NUMBERS**
6. AUTHOR(S) John A. Heropoulos	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.	**12b. DISTRIBUTION CODE** A

13. ABSTRACT (maximum 200 words)

In the past few years, IBM has expanded their Quantum Experience to include a high-level Python module called Qiskit as well as a 15-qubit machine (ibmq_16_melbourne). In order to evaluate IBM's progress, we create an overview of the processes involved in developing and running an algorithm on IBM's quantum machines. This includes exploring the new resources that IBM has made available recently including the Qiskit Python module. Then, we test textbook algorithms such as the Deutsch-Jozsa algorithm, Grover's search algorithm, and Quantum Fourier Transform on ibmq_16_melbourne and compare the results with the output of a simulator. Constraints imposed by IBM's Quantum Experience and significant differences between algorithm performance on IBM's quantum hardware and the simulator will be used to assess IBM's quantum computing capabilities.

14. SUBJECT TERMS Quantum Computing, Qskit, qubit, Deutsch-Joza, Shor's, Grover's, Qexperience, IBM	**15. NUMBER OF PAGES** 67
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89) Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

EVALUATION OF A PHYSICAL QUANTUM COMPUTER

John A. Heropoulos
Ensign, United States Navy
BS, U.S. Naval Academy, 2019

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 2020

Approved by: Theodore D. Huffmire
Advisor

Francesco A. Narducci
Second Reader

Peter J. Denning
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

In the past few years, IBM has expanded their Quantum Experience to include a high-level Python module called Qiskit as well as a 15-qubit machine (ibmq_16_melbourne). In order to evaluate IBM’s progress, we create an overview of the processes involved in developing and running an algorithm on IBM’s quantum machines. This includes exploring the new resources that IBM has made available recently including the Qiskit Python module. Then, we test textbook algorithms such as the Deutsch-Jozsa algorithm, Grover’s search algorithm, and Quantum Fourier Transform on ibmq_16_melbourne and compare the results with the output of a simulator. Constraints imposed by IBM’s Quantum Experience and significant differences between algorithm performance on IBM's quantum hardware and the simulator will be used to assess IBM’s quantum computing capabilities.

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

List of Figures

THIS PAGE INTENTIONALLY LEFT BLANK

List of Tables

THIS PAGE INTENTIONALLY LEFT BLANK

List of Acronyms and Abbreviations

DOD	Department of Defense
NISQ	Noisy Intermediate-Scale Quantum
NPS	Naval Postgraduate School
QASM	Quantum Assembly Language
QFT	Quantum Fourier Transform
QKD	Quantum Key Distribution

THIS PAGE INTENTIONALLY LEFT BLANK

Acknowledgments

I would like to thank Dr. Ted Huffmire for the many hours he dedicated to perfecting this thesis and teaching me about quantum computing.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1: Introduction

While speedup in classical computing comes primary from leveraging various forms of classical parallelism (e.g., instruction-level parallelism, thread-level parallelism, request-level parallelism, etc.), quantum computing leverages quantum parallelism, which evaluates a function on all possible inputs simultaneously. One significant area where a large-scale, fault-tolerant quantum computer would outperform a classical computer is cracking asymmetric cryptography, which relies on the computational time complexity of factoring large integers. Considerable difficulty exists in building a quantum computer due to the challenge of controlling subatomic particles. Current quantum computers are costly and are only capable of factoring small numbers; factoring a large integer is a long-term research challenge. Nevertheless, several companies are developing so-called Noisy Intermediate-Scale Quantum (NISQ) devices as a steppingstone towards this much greater and longer-term challenge and are making some of their machines available to the public [1]. In 2019, Google claimed that it had achieved quantum supremacy [2]. Quantum Supremacy[1] is the point at which a quantum computer can perform a task faster than any classical computer. Google achieved this feat when it used a 53-qubit processor to sample the output of a pseudo-random number circuit [5]. In 2016, IBM launched its *Quantum Experience* which originally offered a 5-qubit machine available via the cloud. The programmer's interface to the 5-qubit machine is both IBM's Quantum Assembly Language (*QASM*) and *Composer*, their graphical user interface. In the past few years, IBM has expanded their *Quantum Experience* to include a high-level Python module called *Qiskit* as well as multiple other 5-qubit machines and a 16-qubit machine (*melbourne*) [6], [7]. Other companies which have made their hardware available for public use are D-Wave and Rigetti. Google provides access to their 3D Quantum Playground which features a simulator and a scripting language. Similarly, Microsoft provides a high-level language, *Q#*, and a simulator [8]. IBM, Rigetti, and Google base their

[1] An alternate term for quantum supremacy is quantum singularity [3], [4]. The task performed need not be a useful one. Achieving quantum supremacy merely means that a quantum computer is capable of performing a specific problem that is infeasible to calculate on a classical computer. It does not mean that quantum computers outperform classical computers on all problems or that the chosen problem has any practical application. In general, future large-scale, fault-tolerant quantum computers will outperform classical computers on a limited set of specific tasks like factoring; the key questions are how much faster and at what cost.

quantum computers on the quantum circuit model [9]. D-wave uses quantum annealing, and Microsoft is attempting to experimentally create what Microsoft believes is a superior qubit. This paper will focus on IBM's quantum computing services, as they form the most intuitive, usable, straightforward, and well-documented ecosystem we have seen thus far. We will investigate the performance of IBM's hardware by running textbook quantum algorithms found in *Quantum Computing For Computer Scientists* by Yanovsky and comparing the result to a quantum simulator.

Because of the global nature of research in quantum information science, it is important for the DoD to stay ahead in its understanding of the current state of quantum computing. For example, China spends an increasing amount on quantum computing R&D [10]. Currently, China is completing a satellite system that relays qubits for secure communication [10]. While Quantum Key Distribution (QKD) [11] is a vastly different, more established, and more mature technology in comparison to quantum computing, it is essential to closely follow international developments in the full portfolio of quantum technology under the vast umbrella of quantum information science, which includes both quantum computing and QKD.

CHAPTER 2:
IBM Quantum Experience

2.1 Background

In 2016, IBM launched its *Quantum Experience*, a cloud-based platform for users to interface with IBM's quantum machines [6]. At the time of writing, the IBM Quantum Experience gives users access to nine devices, including a 32-bit simulator, a 1-qubit machine, six 5-qubit machines, and a 15-qubit machine. The number of quantum experiments performed using the *Quantum Experience* has risen from 100,000 within a month after its release to over 10 million experiments to date [6], [7]. In order to use the *Quantum Experience* users must create an account with IBM. Once the user has an account, they can install software locally or access the *Quantum Experience* through the web-based GUI. Users can access IBM's physical quantum computers through multiple interfaces including the circuit *Composer* and scripts written in either IBM's *Open Quantum Assembly* (*QASM*) language or the Python module *Qiskit*. The *Quantum Experience* also contains resources and support for users in the form of tutorials, a Slack Workspace, user guides, and *Qiskit Textbook*.

2.2 Quantum Assembly Language (QASM)

QASM is a low-level programming language for quantum computers. The language draws on both C and assembly [12]. Quantum circuits specified using *Qiskit* or *Composer* are ultimately compiled to *QASM* before running on the target IBM device. A detailed description of *QASM* can be found in [12], and examples can be found in [13].

2.3 Composer

The *Composer* is a GUI for building circuits. The user can build a circuit by dragging and dropping gates onto a blank circuit or by importing a circuit which has been specified using *QASM*. Circuits can be run from *Composer*, and results can be downloaded as *json* files. In 2019, features added to the *Composer* include showing real-time changes in the simulated quantum states as the circuit is changed as well as real-time changes in the circuit

as changes are made in *Composer's* text editor [7]. This paper will only use *Qiskit* to interface with IBM's quantum computers, but example uses of *Composer* of can be found in the "Resources" section of the *Quantum Experience* web page.

2.4 Qiskit

Qiskit is a Python module released in 2017 which allows users to write scripts in Python to interface with IBM's quantum computers [14]. Qiskit is made up of four different subpackages, *Terra*, *Aer*, *Aqua*, and *Ignis*. *Terra* is the base of *Qiskit* and provides the general tools for users to create and run quantum circuits without too much background knowledge. *Aer* provides users with tools for simulation and modeling of quantum computers. *Aqua* is the module for quantum algorithms and applications ranging from finance to chemistry. *Ignis* can be used for working closer to quantum hardware for purposes such as quantum error correction. *Qiskit* is the method this paper uses to interface with IBM's quantum computers, and the specific subpackages used are *Terra*, *Aer*, and *Aqua*.

CHAPTER 3:
Quantum Computing Overview

3.1 Physics Background

In quantum physics, a system is represented by a wave function, $|\Psi\rangle$. The general equation for the wave function is shown in Equation 3.1.

$$|\Psi\rangle = \sum_{i=0}^{\infty} c_i \, |\psi_i\rangle \tag{3.1}$$

where each $|\psi_i\rangle$ is a possible state of the system, and each c_i is probability amplitude associated with state $|\psi_i\rangle$. The square of a coefficient, $|c_i|^2$, is the probability of finding the particle in state $|\psi_i\rangle$. Because the squared coefficients are probabilities, the sum of the squared coefficients must add up to 1 as shown in Equation 3.2.

$$\sum_{i=0}^{n} |c_i|^2 = 1 \tag{3.2}$$

A single qubit must exist in some combination of states $|0\rangle$ and $|1\rangle$, making a single qubit a two-state system. For n qubits, there are 2^n possible states. When a single qubit is actually measured, it will collapse to either $|0\rangle$ or $|1\rangle$ according to the laws of probability. Consider as an example, a 3-qubit wavefunction given by $|\Psi\rangle = \frac{1}{\sqrt{2}} |000\rangle + \frac{1}{\sqrt{2}} |100\rangle$ whose basis states $|\psi_i\rangle$ and probability amplitudes are tabulated in Table 3.1. In this specific case, there is a 50% probability of finding all particles in state $|0\rangle$ (i.e., $|000\rangle$). There is also a 50% probability of measuring the first qubit in state $|1\rangle$ and the second two qubits in state $|0\rangle$ (i.e., $|100\rangle$).

State	Coefficient
000	.707
001	0
010	0
011	0
100	.707
101	0
110	0
111	0

Table 3.1. The states and coefficients of a 3-qubit wave function. 50% of the time, the wave function will be measured in state $|000\rangle$, and 50% of the time the wave function will be measured in state $|100\rangle$.

The wave function can also be represented as a 1-D matrix of coefficients. Figure 3.1 shows the matrix representation of the wave function from Table 3.1.

$$|\Psi\rangle = \begin{bmatrix} .707 \\ 0 \\ 0 \\ 0 \\ .707 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Figure 3.1. Matrix representation of wave function from Table 3.1

3.2 Qubits

A qubit is the most basic unit of information in quantum computing. In *Qiskit*, a qubit by default is labeled q_i, where i is the order in which the qubits were declared (the first qubit is q_0, the second is q_1, and the nth qubit is q_{n-1}). If only one qubit is used in a circuit, the qubit is labeled q. One way of visualizing qubits in *Qiskit* is the Bloch sphere. The Bloch sphere is a spherical representation of a single-qubit state governed by Equation 3.3 [15].

$$|\psi\rangle = \cos(\theta/2)\,|0\rangle + e^{i\phi}\sin(\theta/2)\,|1\rangle \tag{3.3}$$

$$0 \leq \theta \leq \pi$$

$$0 \leq \phi < 2\pi$$

Figures 3.2 and 3.3 show a simple circuit with one qubit and the corresponding Bloch sphere. A Hadamard gate is applied to q, putting it into an equal superposition of states $|0\rangle$ and $|1\rangle$ assuming an initial state of either purely $|0\rangle$ or purely $|1\rangle$.

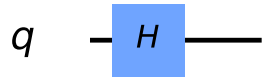

Figure 3.2. A circuit with Hadamard gate applied to qubit q.

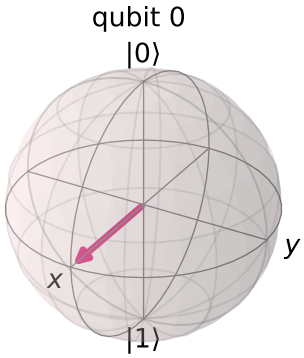

Figure 3.3. A Bloch sphere for the qubit q in Figure 3.2. Drawn using *plot_bloch_vector* function in *Qiskit*.

A significant difference between Qiskit and quantum computing textbooks such as Yanofsky [15] and Nielsen [16] is qubit ordering, which has implications for multi-qubit gates. As explained in the documentation for *Qiskit Aer*:

> When representing the state of a multi-qubit system, the tensor order used in Qiskit is different than that used in most physics textbooks. Suppose there are n qubits, and qubit j is labeled as q_j. *Qiskit* uses an ordering in which the nth qubit is on the left side of the tensor product, so that the basis vectors are labeled as $q_n \ldots \otimes q_1 \otimes q_0$. For example, if qubit zero is in state $|0\rangle$, qubit 1 is in state $|0\rangle$, and qubit 2 is in state $|1\rangle$, Qiskit would represent this state as $|100\rangle$ whereas many physics textbooks would represent it as $|001\rangle$. This difference in labeling affects the way multi-qubit operations are represented as matrices. [17]

3.3 Gates

Quantum gates are primitive operations that transform the quantum states of the quantum bits to which they are applied. With the exception of measurement gates, all quantum gates must be reversible due to thermodynamic considerations [18]. Quantum gates are represented by unitary matrices [17]. *Qiskit* offers a wide range of gates that are commonly used in quantum computing. These gates can be found in the *Qiskit* tutorial. Although *Qiskit*

will display a circuit with the gates that the user specifies, the circuit composed by the user is ultimately compiled to a minimized circuit that is optimized for the specific quantum processor architecture. IBM calls this compilation procedure *transpilation*. The gates that a given IBM machine uses can be found by viewing its status on the *Qexperience* homepage. The gates used by the IBM machine are shown in Figure 3.4 under the section titled "Basis Gates."

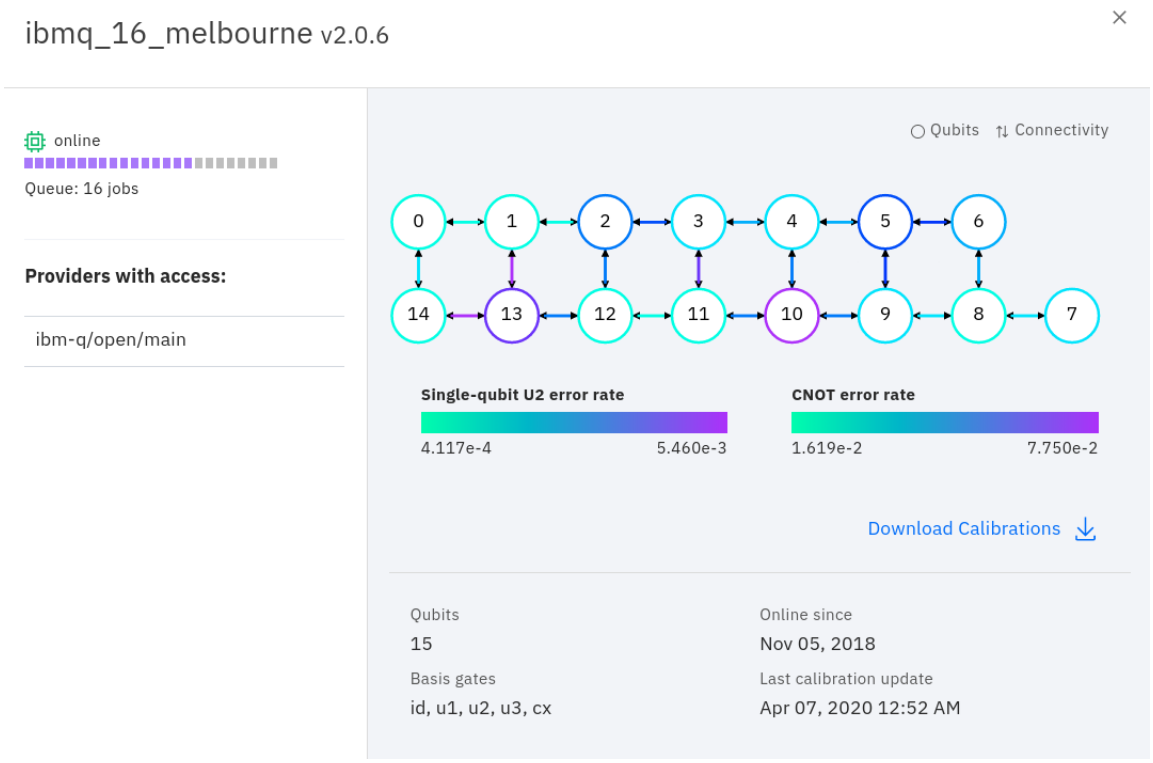

Figure 3.4. *ibmq_16_melbourne* status accessed from IBM Qexperience homepage. The U_1, U_2, and U_3 gates are IBM's generic gates. id refers to the identity gate, and cx is a controlled-not gate.

To see a circuit after it is compiled into the basis gates of the IBM machine, one can use the *Qiskit* transpiler. The *transpile* function is located in the *Qiskit.compile* module. Figure 3.5 shows a simple circuit consisting of a Hadamard gate and a measurement gate. Figure 3.6 shows the transpiled version of the circuit in Figure 3.5. Further information on transpilation can be found in Appendix B.

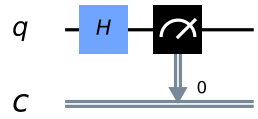

Figure 3.5. A simple circuit.

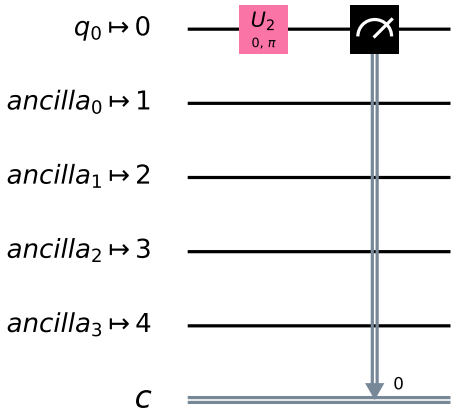

Figure 3.6. Transpiled version of the simple circuit in 3.5. Ancilla qubits are constant qubits used in simple operations.

The U_2 gate in Figure 3.6 is one of the basis gates on the IBM machine. The most general of the U gates is the U_3 gate. The U_1 and U_2 gates can be described in terms of the U_3 gate. The general matrix representation for the U_3 gate is shown in Equation 3.4.

$$U_3 = \begin{bmatrix} \cos\theta/2 & -e^{i\lambda}\sin\theta/2 \\ e^{i\phi}\sin\theta/2 & e^{i\phi+i\lambda}\cos\theta/2 \end{bmatrix} \tag{3.4}$$

The U_3 gate parameters θ, ϕ, and λ can be adjusted to form any unitary matrix. The U_2 gate can be used to create a superposition of states, and the U_1 gate can only be used to

change the phase of a state [17]. The U_2 and U_1 gates in terms of the U_3 gate are shown in Equations 3.5 and 3.6.

$$U_2 = U_3(\pi/2, \phi, \lambda) \tag{3.5}$$

$$U_1 = U_3(0, 0, \lambda) \tag{3.6}$$

In the case of multi-qubit gates, IBM's matrix representation is different from textbooks. To illustrate this difference, we use the simple circuit shown in Figure 3.7.

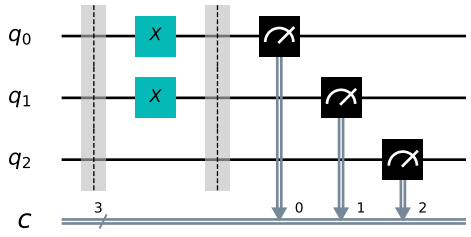

Figure 3.7. A circuit with a NOT gate applied to each of the top two qubits.

Using the textbook method, the matrix representation of the region between the barriers is calculated as $NOT \otimes NOT \otimes I$. Using the *Qiskit* method, the corresponding matrix is calculated as $I \otimes NOT \otimes NOT$. The difference between these two matrices is shown in Figures 3.8 and 3.9.

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure 3.8. Textbook matrix.

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

Figure 3.9. Qiskit matrix.

3.4 Oracles

A quantum oracle is a black-box operation that has some relation to the function that an algorithm is attempting to investigate. Oracles do not know the answer to the problem a circuit attempts to solve, but oracles do produce some output which helps the algorithm converge on a solution. In [15] and [16], the circuit element representing the oracle is labeled U_f. *Qiskit* provides oracles for several types of functions, including truth tables, logical expressions, and custom user functions. A truth table oracle is constructed from one or more bitmaps. The truth table in Table 3.2 corresponds to the bitmap 0110 (the XOR function).

x	y	x XOR y
0	0	0
0	1	1
1	0	1
1	1	0

Table 3.2. Truth Table for two-qubit XOR.

Logical expression oracles can be defined in terms of a string. For example, "x | (y ^ z)" is a logical expression over three variables. *Qiskit* creates the circuit for the oracle automatically so users do not need to construct the U_f gate. Figure 3.10 shows the circuit element formed from a *TruthTableOracle* object.

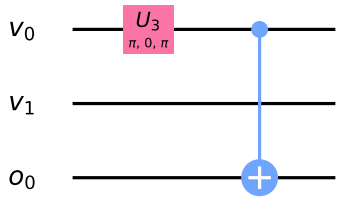

Figure 3.10. Circuit generated by *TruthTableOracle* for the XOR bitmap.

3.5 Entanglement

In an entangled system of particles, a measurement on one particle influences the state of another particle, irrespective of the distance between them [15]. Many applications of quantum computing rely heavily on quantum entanglement [16]. The Bell States are an important set of four entangled states. If a system of two qubits exists in the Bell 00 (β_{00}) State and one of the qubits is measured as being in a particular state ($|0\rangle$ or $|1\rangle$), the other qubit must be in the same state. The β_{00} state is shown in Equation 3.7.

$$|\beta_{00}\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) \tag{3.7}$$

On IBM's machine, we can create the β_{00} state by using the circuit shown in Figure 3.11.

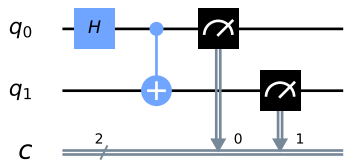

Figure 3.11. Entanglement circuit in *Qiskit* to create the β_{00} state and then collapse it through measurement.

When the circuit is run on IBM's machine, the measurements should only yield $|00\rangle$ and $|11\rangle$ in approximately equal proportion. Running the entanglement circuit on the IBM machine yields the expected result as shown in Figure 3.12.

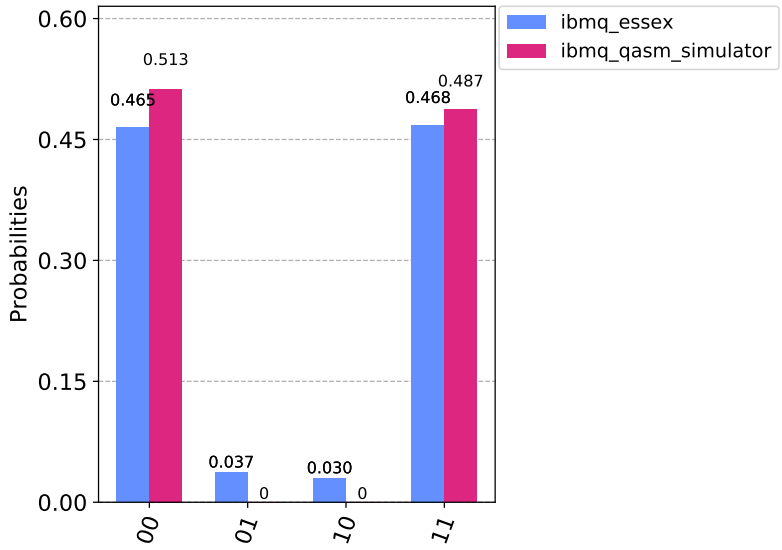

Figure 3.12. Results of running entanglement circuit on *ibmq_ourense*. Quantum noise causes the system to exist in states $|01\rangle$ and $|10\rangle$ a small percent of the time.

3.6 Teleportation

Quantum teleportation is the process of moving the state of a qubit to another location while destroying the original state. Teleportation can be used to move the state of a qubit over an arbitrary distance. The teleportation circuit builds on the entanglement circuit by adding another qubit and a few gates. An explanation of the exact steps of the teleportation process can be found in [15]. The teleportation circuit is shown in Figure 3.13.

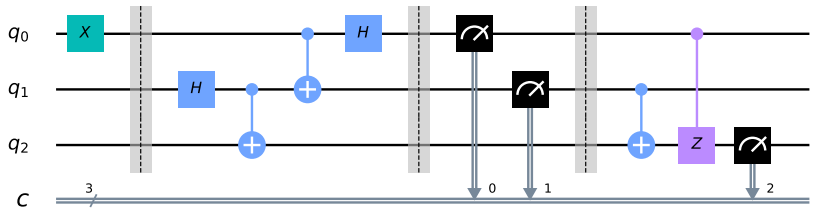

Figure 3.13. Teleportation Circuit. The top qubit q_0 is $|1\rangle$ following the X-gate. The state of the top qubit q_0 is teleported to the bottom qubit q_2.

Because measurement affects the state of a qubit, the teleportation circuit cannot be run the way it is shown in Figure 3.13. Instead, the measurement gates for the top two qubits must be moved to the end of the circuit with the other measurement gate. According to IBM, postponing measurement is allowable under the *deferred measurement principle*. IBM states that "any measurement can be postponed until the end of the circuit: We can move all the measurements to the end, and we should see the same results" [17].

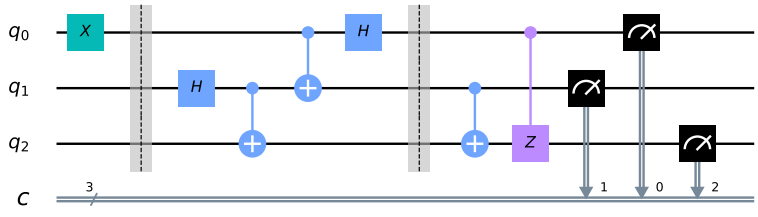

Figure 3.14. Teleportation Circuit with deferred measurement gates for the top two qubits.

Figure 3.15 shows the result of running the Teleportation Circuit on the IBM machine.

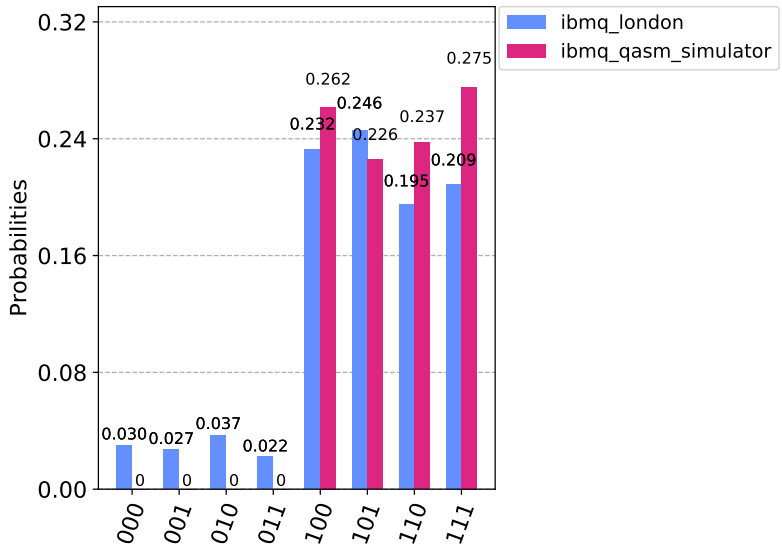

Figure 3.15. Result of running the teleportation circuit shown in 3.14 on *ibmq_london*. The bottom qubit is measured in state $|1\rangle$ most of the time with some error due to noise.

CHAPTER 4:
Algorithms

4.1 Deutsch-Jozsa

The purpose of the Deutsch-Jozsa algorithm is to determine whether a function is constant or balanced. Given a function $f : \{0,1\}^n \rightarrow \{0,1\}$, f is balanced if half of the inputs map to 0, and the other half map to 1, as shown in Figure 4.1. Function f is constant if all inputs map to 0, or all map to 1, as shown in Figure 4.2. On a classical computer, the Deutsch-Jozsa algorithm requires evaluating all $2^n + 1$ possible inputs, while on a quantum computer, it can determine whether a function is constant or balanced in a single function evaluation. [15].

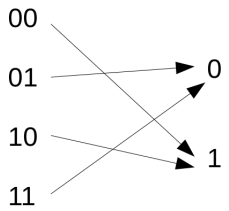

Figure 4.1. A balanced function which outputs 0 for half of all inputs and 1 for the other half.

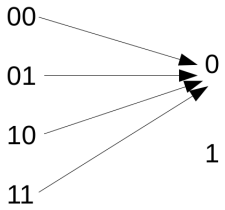

Figure 4.2. A constant function which outputs 0 for all inputs.

The Deutsch-Jozsa algorithm does not return meaningful information about a function which is neither constant nor balanced. Although the Deutsch-Jozsa algorithm is an academic

example, it provides a good illustration of quantum parallelism. Using *Qiskit*, one can either build a Deutsch-Jozsa circuit from scratch or use the built-in Deutsch-Jozsa algorithm in the *Qiskit Aqua* module. The argument to the constructor for the *DeutschJozsa* class object in *Qiskit Aqua* is a TruthTableOracle object [17]. A quantum oracle is a way of encoding a quantum function which is then used to construct a circuit with gates forming the U_f specific to that function. The circuit shown in Figure 4.3 shows the Deutsch-Jozsa algorithm run on a function with the output sequence '1010'. The U_3 gate and the Toffoli gate form the U_f for the circuit.

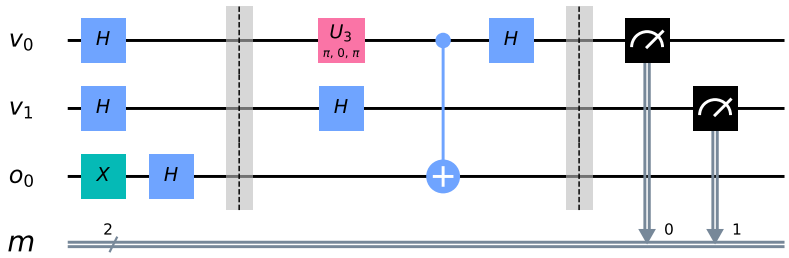

Figure 4.3. Deutsch-Jozsa circuit in *Qiskit* for a function with output sequence "1010"

The top two qubits are measured in the third step of the circuit with a measurement of 00 corresponding to the function being constant, and any other measurements corresponding to the function being balanced.

This paper uses the Python scientific computing package *Numpy* to simulate various circuits using matrices to ensure circuit correctness and compare textbook and IBM frameworks [19]. In order to ensure that IBM's circuit is correct, the circuit is simulated in *Numpy* using matrices. The experiment is run using both IBM's qubit ordering and representation of U_f as well as the textbook qubit ordering and the textbook U_f. In order to show the equivalence of the textbook and the IBM U_f matrices, the experiment is repeated, using a hybrid approach, where the IBM U_f matrix is used in the textbook circuit using textbook qubit ordering. The difference between the IBM matrix and textbook matrix is highlighted by Figures 4.4 and 4.5.

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 4.4. IBM U_f matrix with textbook qubit ordering

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 4.5. Textbook U_f matrix with textbook qubit ordering

Despite the differences in these matrices, they both yield the same result; specifically, that the function is balanced. Note that the IBM measurement still yields $|01\rangle$ for the top two qubits, but this is okay as long as they agree for constant functions. The results of using different combinations of IBM and textbook matrices are shown in Tables 4.1 and 4.2.

State	IBM	IBM with textbook Uf	Textbook
000	0	0	0
001	.707	0	0
010	0	0	.707
011	0	0	.707
100	0	.707	0
101	.707	.707	0
110	0	0	0
111	0	0	0

Table 4.1. Results of running the Deutsch-Jozsa circuit for function with output sequence "1010" using: (1) IBM's U_f with IBM qubit ordering; (2) hybrid approach incorporating IBM's U_f in textbook circuit with textbook qubit ordering; and (3) textbook U_f with textbook qubit ordering. All combinations correctly show that the function is balanced.

In order to ensure consistency, the experiment is run again, this time on the constant function with output sequence "1111."

State	IBM	IBM with textbook U_f	Textbook
000	.707	.707	.707
001	0	.707	.707
010	0	0	0
011	0	0	0
100	.707	0	0
101	0	0	0
110	0	0	0
111	0	0	0

Table 4.2. Results of running the Deutsch-Jozsa circuit for function with output sequence "1111" using: (1) IBM's U_f with IBM qubit ordering; (2) hybrid approach incorporating IBM's U_f in textbook circuit with textbook qubit ordering; and (3) textbook U_f with textbook qubit ordering. All combinations correctly state that the function is constant with the relevant qubits in state $|00\rangle$.

Next, IBM's Deutsch-Jozsa algorithm implementation is tested on *ibmq_essex*, one of IBM's 5-qubit quantum computers. The results of four different circuits on *ibmq_essex* are shown alongside the results from IBM's simulator in Figures 4.6, 4.7, 4.8, and 4.9. The first function used is the one with output sequence "1010." While *ibmq_essex* still arrives at the correct answer in Figure 4.8, variation in output probabilities exists between *ibmq_essex* and the ideal simulator due to quantum noise.

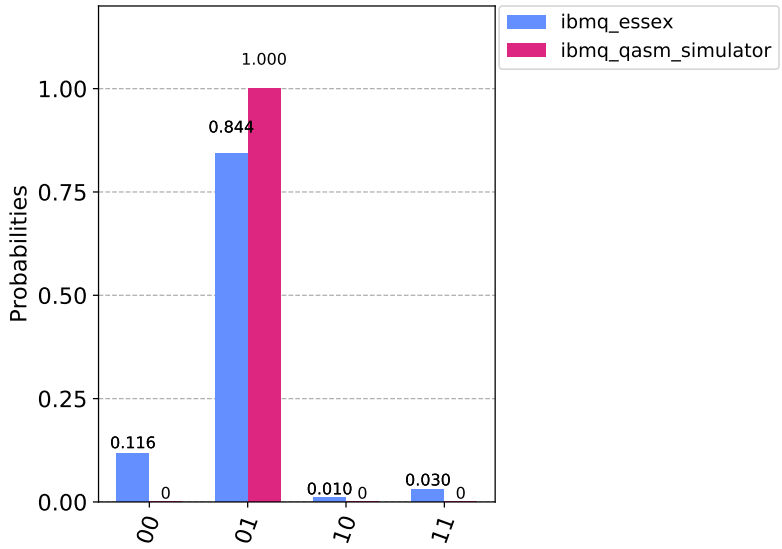

Figure 4.6. Results of running Deutsch-Jozsa circuit for function with output sequence "1010" on both *ibmq_essex* and IBM's ideal simulator

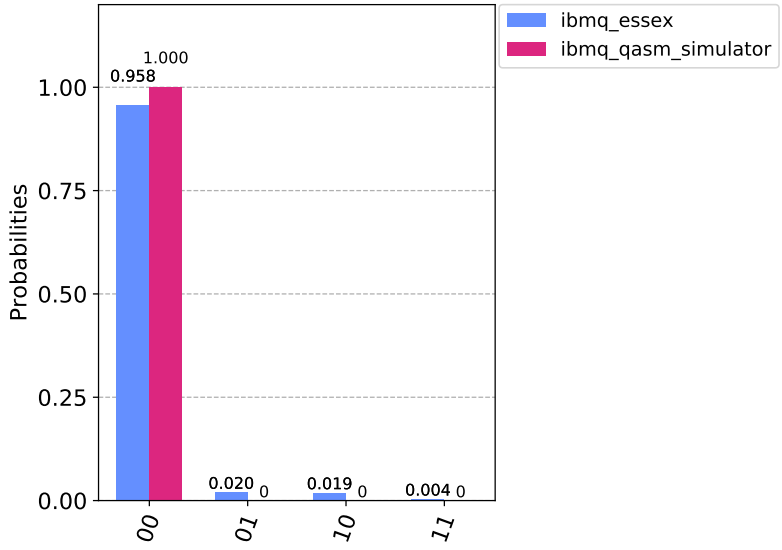

Figure 4.7. Results of running Deutsch-Jozsa circuit for function with output sequence "1111" on both *ibmq_essex* and IBM's ideal simulator

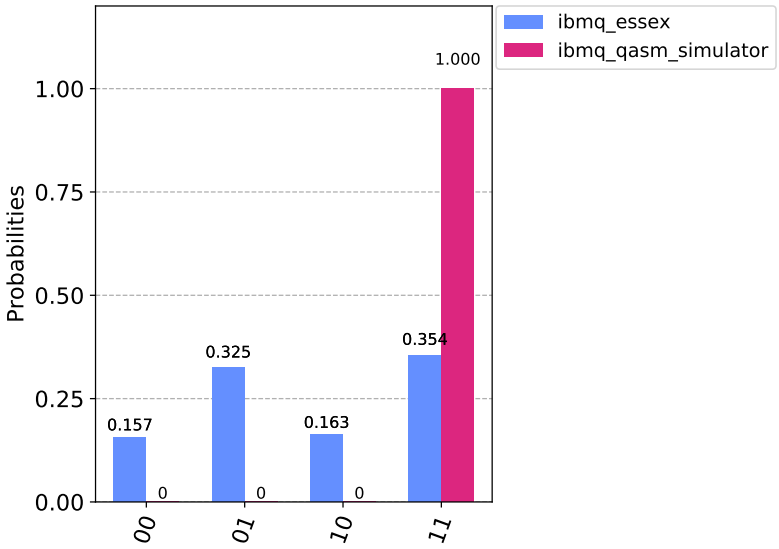

Figure 4.8. Results of running Deutsch-Jozsa circuit for function with output sequence "1001" on both *ibmq_essex* and IBM's ideal simulator.

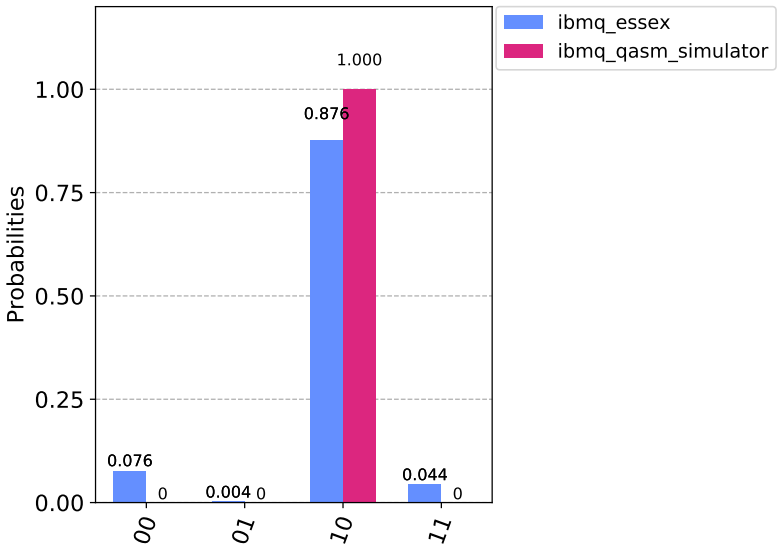

Figure 4.9. Results of running Deutsch-Jozsa circuit for function with output sequence "1100" on both *ibmq_essex* and IBM's ideal simulator

4.2 Grover's Algorithm

The purpose of Grover's algorithm is to search for an item in an unordered array. More formally, given a function $f : \{0,1\}^n \rightarrow \{0,1\}$, find a string x_0 for which $f(x = x_0) = 1$ and $f(x \neq x_0) = 0$. Grover's algorithm provides $O(\sqrt{n})$ speedup over classical search [15], [16]. Due to this speedup, Grover's algorithm is especially applicable to database searches. Similar to the Deutsch-Jozsa algorithm in *Qiskit*, a circuit for Grover's algorithm can either be made from scratch or generated through the built-in *Grover* class object in the *Aqua* module. The *Grover* object takes a quantum oracle as its argument. The circuit for running Grover's algorithm on a 3-qubit function for which $f(000) = 1$ is shown in Figure 4.10. While most physics textbooks would use four qubits for Grover's algorithm on this function, *Qiskit* generates a five-qubit circuit. The reason for this discrepancy is unclear and merits further investigation.

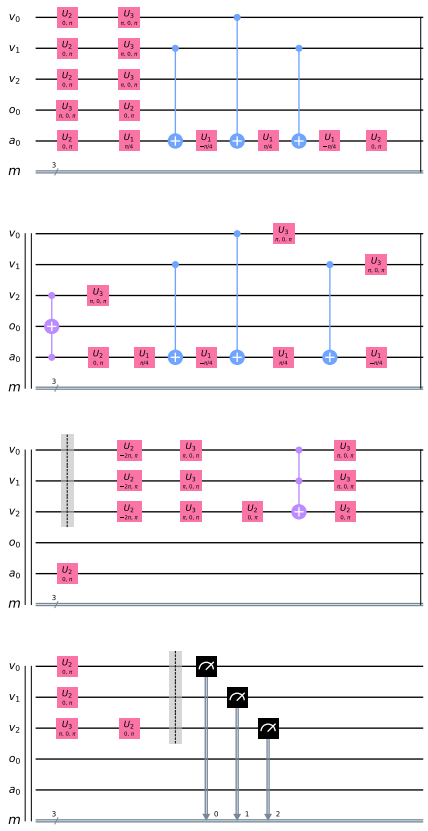

Figure 4.10. Grover circuit in *Qskit* for 3-qubit function with $f(000) = 1$

The first experiment tests the textbook and IBM results for a 3-qubit function f(000) = 1. Running the textbook circuit using *Numpy* matrices yields the result shown in Table 4.3.

State	Coefficient
0000	.688
0001	.688
0010	.062
0011	.062
0100	.062
0101	.062
0110	.062
0111	.062
1000	.062
1001	.062
1010	.062
1011	.062
1100	.062
1101	.062
1110	.062
1111	.062

Table 4.3. Result of running the textbook circuit for 3-qubit function with $f(000) = 1$ using matrices in *Numpy*.

The results of running the IBM circuit on the IBM machine are shown in Figure 4.11.

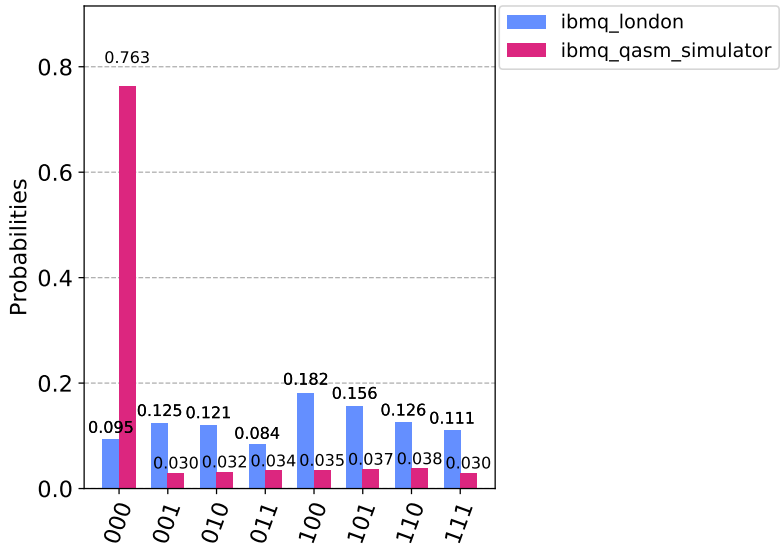

Figure 4.11. Result of running the IBM Grover circuit for 3-qubit function with $f(000) = 1$ on *ibmq_london*.

As expected, the state with the largest magnitude in Figure 4.11 is the $|000\rangle$ state. The experiment is repeated on the IBM machine with several other bit sequences. The results are shown in Figures 4.11, 4.12, and 4.13.

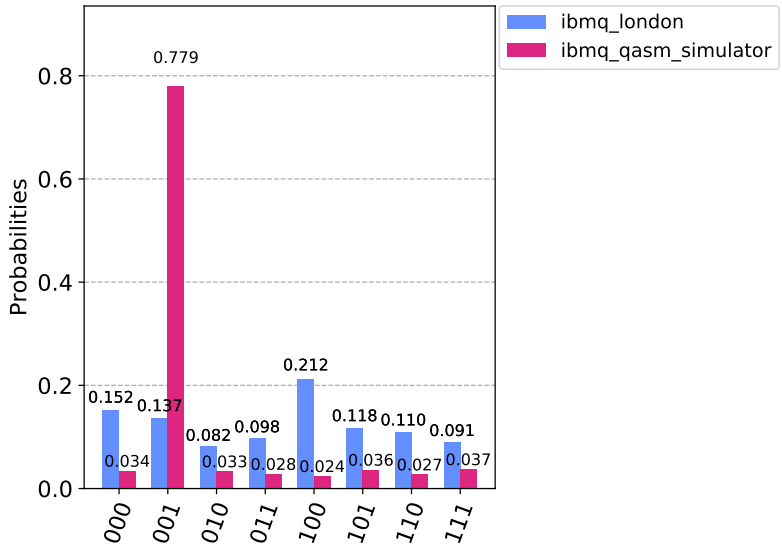

Figure 4.12. Result of running the IBM Grover circuit for 3-qubit function with $f(001) = 1$ on *ibmq_london*.

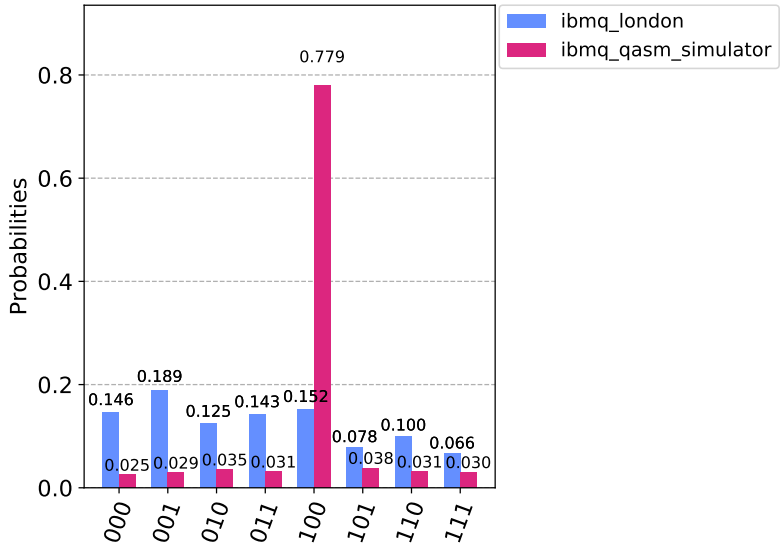

Figure 4.13. Result of running the IBM Grover circuit for 3-qubit function with $f(100) = 1$ on *ibmq_london*.

As shown in Figures 4.11, 4.12, and 4.13, the IBM machine does not always produce the correct maximum amplitude state due to quantum noise.

4.3 Quantum Fourier Transform

The Discrete Fourier Transform (DFT) decomposes a signal or function into its constituent frequencies. The DFT is important in science and engineering. The Quantum Fourier Transform (QFT) is a method of running a DFT on a quantum computer. The fastest algorithm for computing a DFT on a classical computer is the Fast-Fourier Transform algorithm (FFT) which runs in $O(n \log n)$ time. The QFT runs in $O(n)$ time [15]. The QFT is a critical element of Shor's algorithm, as it amplifies the desired period so that there is a very high probability of measuring it at the end of the quantum portion of the algorithm; in the subsequent classical portion of the algorithm, the factors are computed from the period [15]. In *Qiskit*, one can construct his or her own QFT circuit or use the *FourierTransformCircuits* object, which is part of the *Aqua* module. The *construct_circuit* method of the *FourierTransformCircuits* object takes a circuit and a register of qubits as its argument and performs a QFT on the given register of qubits [17]. Figure 4.14 shows a circuit which performs the 3-qubit QFT.

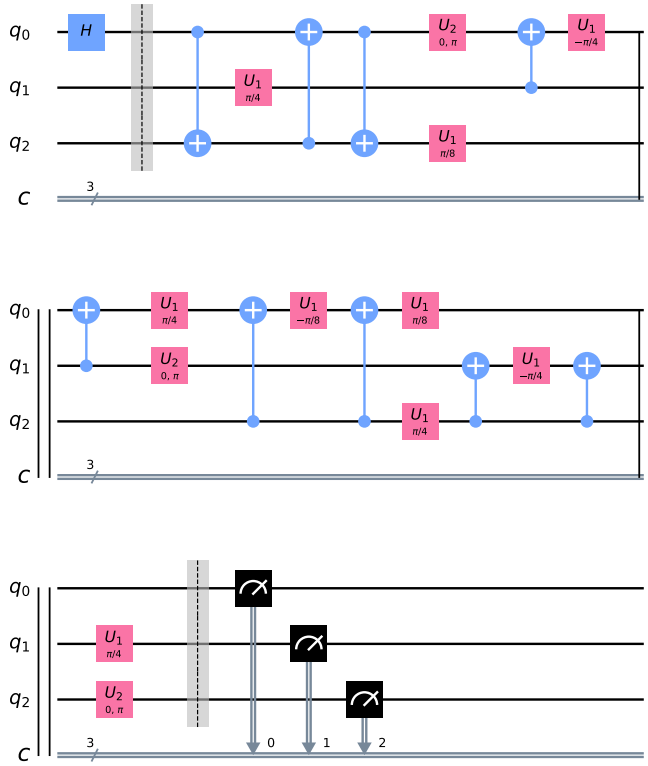

Figure 4.14. *Qiskit* circuit for performing QFT on three qubits.

In order to test the accuracy of the QFT in *Qiskit*, the results of the QFT on the IBM machine are compared to the results of the FFT function in *Numpy*. The experiment is performed on multiple 3-qubit states. The first experiment applies a Hadamard gate to the top qubit and then applies a three-qubit QFT to the three-qubit system. Table 4.4 shows the ideal results obtained using matrix operations and the FFT function in *Numpy*.

State	Coefficient
000	.25
001	.213
010	.125
011	.037
100	0
101	.037
110	.125
111	.213

Table 4.4. Results obtained using matrices in *Numpy*. A Hadamard gate is applied to the top qubit, and then the FFT is applied.

Figure 4.15 shows the results of applying a Hadamard gate to the top qubit and then applying the QFT on the IBM machine.

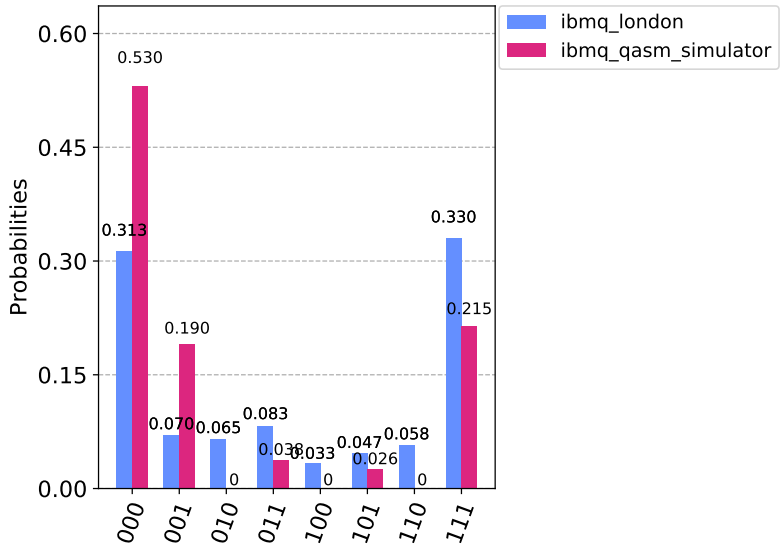

Figure 4.15. Applying a Hadamard gate to the top qubit then applying a QFT in *Qiskit*. Run on *ibmq_london*.

The results on *ibmq_london* are close to the ideal results in Figure 4.15. The $|100\rangle$ state should ideally never be measured according to both the *Numpy* FFT calculation and IBM simulator. The $|100\rangle$ state is measured on *ibmq_london* due to quantum noise. The next experiment consists of applying a Hadamard gate to the top two qubits then applying a 3-qubit QFT to the 3-qubit system. The results are shown in Figure 4.16.

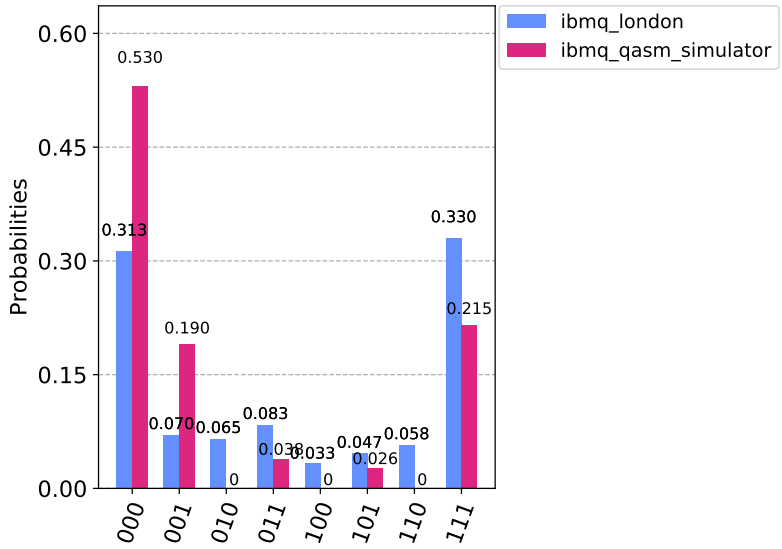

Figure 4.16. Applying a Hadamard gate to the top two qubits then applying a QFT in *Qiskit*. Run *ibmq_london*.

The final experiment applies a Hadamard gate to each of the three qubits, putting them into a superposition state of all $2^3 = 8$ possible values, followed by a three-qubit QFT to the three-qubit quantum system. The resultss are shown in Figure 4.17.

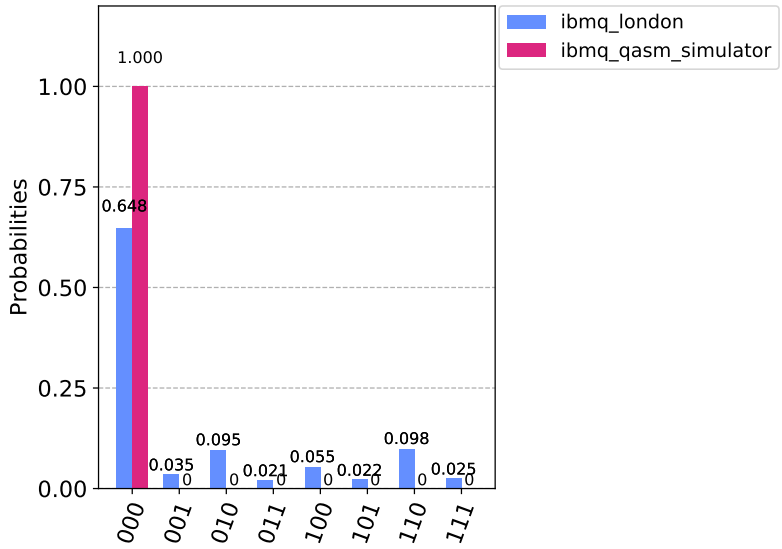

Figure 4.17. Applying a Hadamard gate to all three qubits then applying a QFT in *Qiskit*. Run on *ibmq_london*.

In Figure 4.17, the state $|000\rangle$ should be measured 100% of the time (shown by IBM's ideal simulator). The applications of the DFT in science and engineering rely on the DFT being much more accurate than the results seen on *ibmq_london*.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 5:
Conclusion

Using IBM's *Quantum Experience*, freely available to the public, this thesis ran a variety of quantum algorithms on IBM's quantum processor hardware and reconciled the real-world results against the textbook descriptions of these algorithms. Chapter 2 explained the process of constructing quantum circuits and running quantum algorithms. IBM's *Qiskit* utility facilitates function definition and oracle construction based in IBM's basis of quantum logic gates. This thesis used the truth table and logical expression oracles. More information on how a user can create his or her own oracle can be found in the *Qiskit* documentation [17]. Future work should explore using *Qiskit* to solve problems which require a function that cannot be defined in terms of a truth table or a logical expression. Two fundamental principles, entanglement and teleportation, were both successfully demonstrated on IBM's quantum hardware. In Chapter 3, textbook quantum algorithms were compared to the *Qiskit* implementation. The IBM circuits were simulated using matrices to ensure their correctness. The results of the Deutsch-Jozsa algorithm and Grover's algorithm on IBM's quantum hardware aligned with the ideal results. The results for the QFT were close to the results for the FFT, but the QFT is too noisy to be practical. The algorithms used in this thesis did not incorporate any error correction. Future work should leverage *Qiskit's* quantum error correction capability. *Qiskit* offers additional algorithms for solving problems in finance, chemistry, and optimization. Future work should explore uses of *Qiskit* beyond the suite of algorithms traditionally presented in textbooks. Another interesting study would be to compare IBM's *Quantum Experience* to the cloud-based computing services offered by D-Wave and Rigetti. Appendix A details the results of running the Deutsch-Jozsa algorithm and Grover's algorithm on *ibmq_16_melbourne* using all available 15 qubits. The results suggest that the quantum noise is too great for Grover's algorithm to be accurate on 15 qubits. However, the *ibmq_16_melbourne* experiments also do not use error correction. Cloud-based services like the IBM *Quantum Experience* are bringing the rapidly evolving field of quantum computing to the computer science community.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A:
Running on *Melbourne*

A.1 Background

Melbourne has the most qubits of any machine currently offered to individual users of IBM's *Quantum Experience* at the time of writing this paper. In order to test its performance, the Deutsch-Jozsa algorithm and Grover's algorithm are run on *melbourne's* 15 qubits.

A.2 Deutsch-Jozsa

The first algorithm run is the Deutsch-Jozsa algorithm. The function used corresponds to a bitmap of length $2^{14} = 16384$ zeros. The results are shown in Figure A.1. The most probable state is displayed.

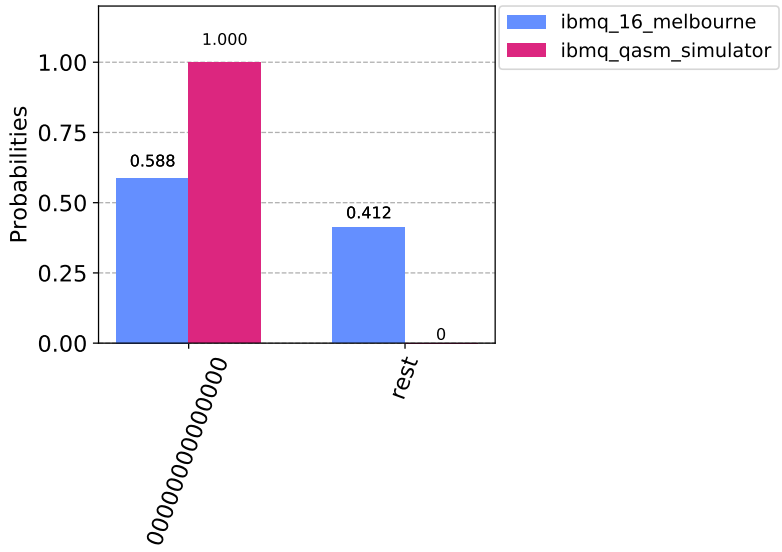

Figure A.1. Results of running Deutsch-Josza algorithm on *ibmq_16_melbourne* for a constant function on 15-qbits.

The results of the 15-qubit Deutsch-Jozsa algorithm on *ibmq_16_melbourne* agree with the results on IBM's simulator as both correctly indicate that the function is constant.

A.3 Grover

Grover's algorithm is performed on a function with a bitmap of length $2^9 = 512$ with f(x = 000000000) = 1 and f(x != 0) = 0. The most probable state (leftmost state in plots) for Grover's algorithm should be 000000000 because f(000000000) = 1. The results on IBM's simulator and *ibmq_16_melbourne* are shown in Figure A.2 and Figure A.3, respectively. The five most probable states for both runs are displayed.

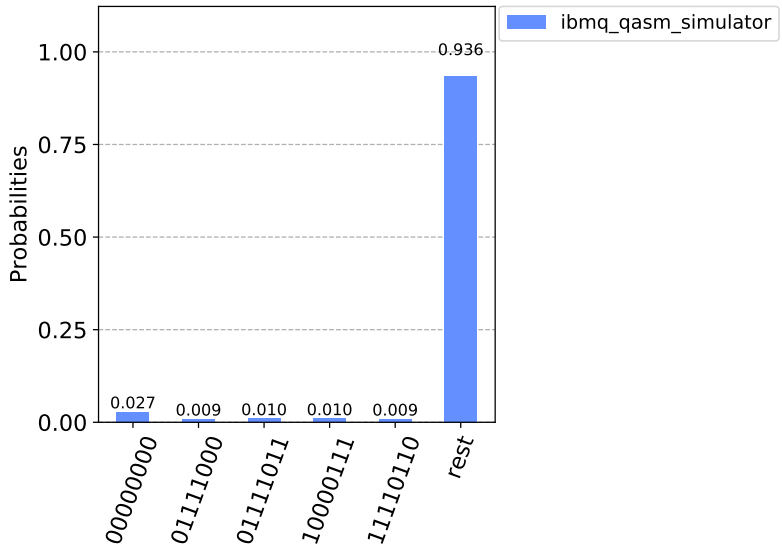

Figure A.2. Results of running 15-qubit Grover's algorithm on IBM's simulator. 000000000 is the most probable state.

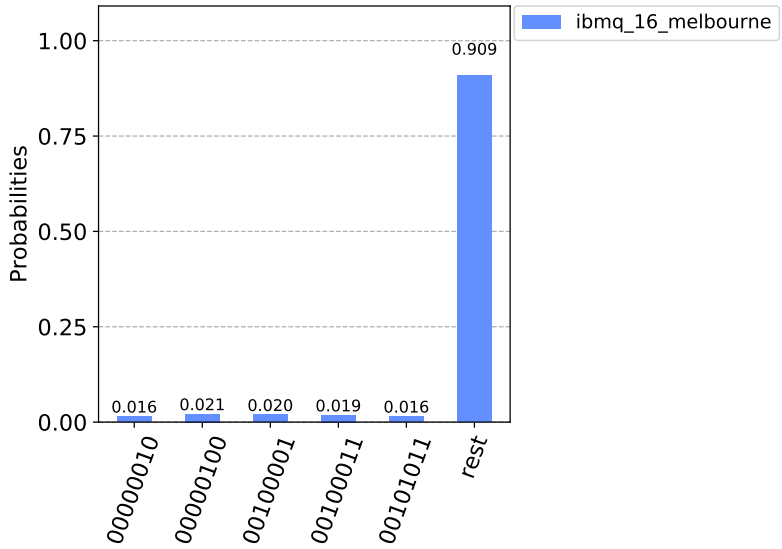

Figure A.3. Results of running 15-qubit Grover's algorithm on *ibmq_16_melbourne*.

For Grover's algorithm, *ibmq_16_melbourne* does not arrive at the correct solution of 00000000 due to quantum noise.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B: Transpilation

Before a user-defined circuit is run on an IBM quantum machine, the circuit is *transpiled* into a circuit which only contains the basis gates of the machine. When *Qiskit* transpiles a circuit, it also optimizes the circuit. In the transpilation process, the user can choose from four levels of optimization from 0 to 3, where 0 is no optimization and 3 is heavy optimization. The default optimization level is 1, light optimization. The greater the level of optimization, the slower the transpilation process [17]. Figures B.1 and B.2 show a transpiled circuit with no optimization (level 0) and light optimization (level 1), respectively.

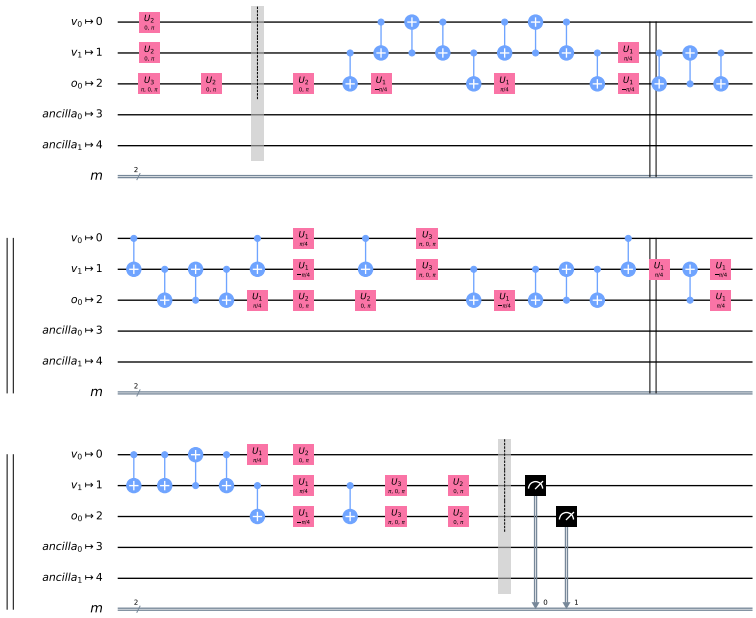

Figure B.1. Transpiled Deutsch-Jozsa circuit for sequence '1001' with level 0 optimization.

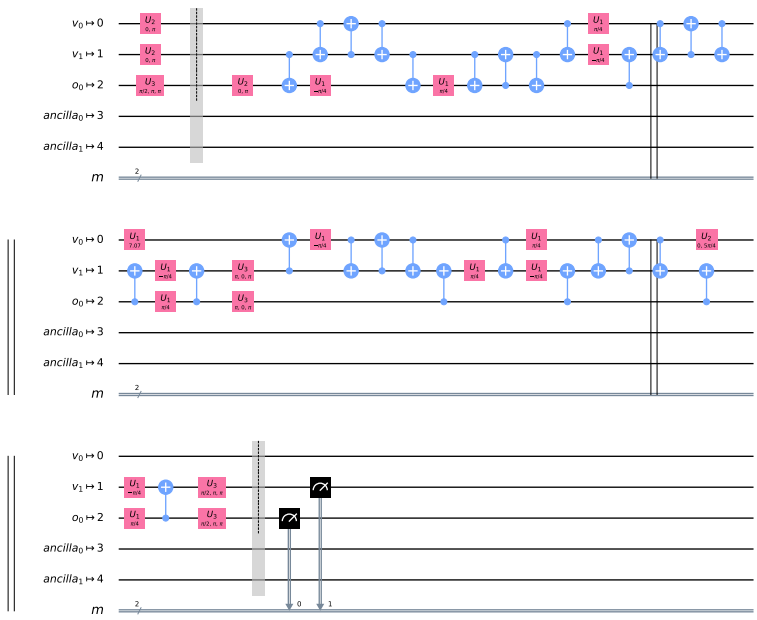

Figure B.2. Transpiled Deutsch-Jozsa circuit for sequence '1001' with level 1 optimization.

Without counting the number of operations in the circuit, one can clearly see that the length of the circuit was reduced by switching from no optimization to light optimization. The size of the circuit for the Deutsch-Jozsa algorithm can also vary with different bit strings. Figures B.3 and B.4 show a simpler Deutsch-Jozsa circuit for bit string '1100' with level 0 and level 1 optimization, respectively.

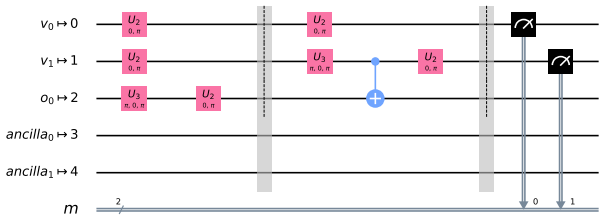

Figure B.3. Transpiled Deutsch-Jozsa circuit for sequence '1100' with level 0 optimization.

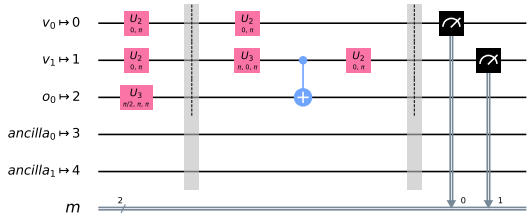

Figure B.4. Transpiled Deutsch-Jozsa circuit for sequence '1100' with level 1 optimization.

The increase in optimization level from 0 to 1 resulted in a reduction of one gate from Figure B.3 to Figure B.4. Even for small circuits, the optimization feature of transpilation can result in a simpler circuit. In order to demonstrate the general effects of *Qiskit's* optimization feature, the various optimization levels are tested on 100 randomly generated 5-qubit circuits and the average number of operations for each level is recorded. The results are shown in B.5.

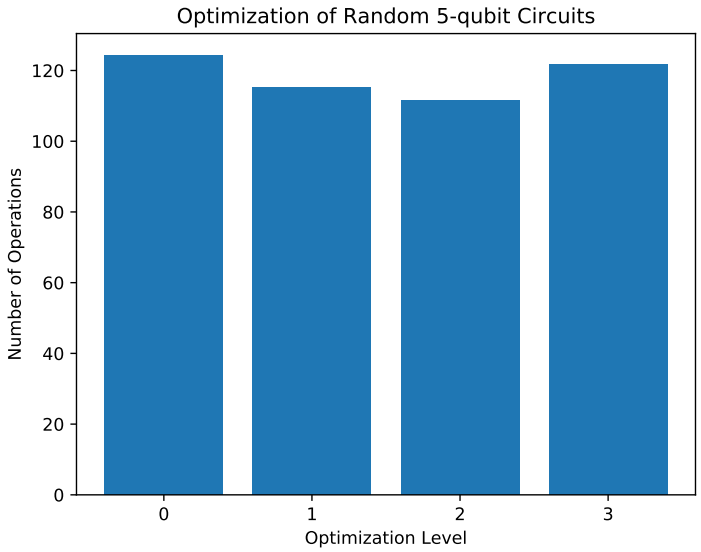

Figure B.5. Average number of operations for a 5-qubit circuit for each optimization level. 100 different random 5-qubit circuits were used for this experiment.

The results shown in B.5 suggest that the increasing the optimization level up to level 2 decreases the average number of operations in 5-qubit circuits.

List of References

[1] J. Preskill, "Quantum Computing in the NISQ era and beyond," *Quantum*, vol. 2, p. 79, Aug. 2018. Available: https://doi.org/10.22331/q-2018-08-06-79

[2] "What we learned in science news in 2019," *The New York Times*, 21 Dec. 2019. Available: https://www.nytimes.com/2019/12/21/science/science-news-2019.html

[3] L. Hardesty, "The quantum singularity," *MIT News*, Mar. 2011. Available: http://news.mit.edu/2011/quantum-experiment-0302

[4] S. Aaronson and A. Arkhipov, "The computational complexity of linear optics," in *Proceedings of the forty-third annual ACM symposium on Theory of computing*, Jun. 2011, pp. 333–342.

[5] F. Arute *et al.*, "Quantum supremacy using a programmable superconducting processor," *Nature*, vol. 574, no. 7779, pp. 505–510, 23 Oct. 2019.

[6] IBM Research Editorial Staff, "The IBM Quantum Experience Gathers Momentum," Blog Post, IBM Research Blog, IBM Corporation, 24 May 2016. Available: https://www.ibm.com/blogs/research/2016/05/ibm-quantum-experience-gathers-momentum-2/

[7] J. Gambetta and J. Chow, "IBM Unveils Beta of Next Generation Quantum Development Platform," Blog Post, IBM Research Blog, IBM Corporation, 8 May 2019. Available: https://www.ibm.com/blogs/research/2019/05/next-gen-ibmqx/

[8] Microsoft. Quantum computing. [Online]. Available: https://www.microsoft.com/en-us/quantum. Downloaded on 20 May 2020.

[9] Rigetti Computing. Rigetti Computing. [Online]. Available: https://rigetti.com/. Downloaded on 20 May 2020.

[10] J. Whalen, "The quantum revolution is coming, and Chinese scientists are at the forefront," *The Washington Post*, 18 Aug. 2019. Available: https://www.washingtonpost.com/business/2019/08/18/quantum-revolution-is-coming-chinese-scientists-are-forefront/

[11] J. Mullins, "Making unbreakable code," *IEEE Spectrum*, vol. 39, no. 5, pp. 40–45, May 2002.

[12] A. W. Cross, L. S. Bishop, J. A. Smolin, and J. M. Gambetta, "Open quantum assembly language," *arXiv preprint arXiv:1707.03429*, pp. 29–36, Jul. 2017.

[13] J. Kasel, "Quantum computing on a physical quantum computer," M.S. Thesis, Naval Postgraduate School, Monterey, CA, Jun. 2018. Available: https://calhoun.nps.edu/handle/10945/59694

[14] Lubensky, David, "Quantum computing gets an API and SDK," Blog Post, The developerWorks Blog, IBM Corporation, 6 Mar. 2017. Available: https://developer.ibm.com/dwblog/2017/quantum-computing-api-sdk-david-lubensky/

[15] N. S. Yanofsky and M. A. Mannucci, *Quantum Computing for Computer Scientists*, 1st ed. New York, NY, USA: Cambridge University Press, 2008.

[16] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information: 10th Anniversary Edition*, 2nd ed. New York, NY, USA: Cambridge University Press, 2011.

[17] H. Abraham *et al.*, "Qiskit: An open-source framework for quantum computing," Jan. 2019. Available: https://doi.org/10.5281/zenodo.2562111

[18] T. Brun, "Quantum Circuits," EE Lecture 11, USC Viterbi School of Engineering, Los Angeles, CA, 14 Feb. 2012. Available: https://viterbi-web.usc.edu/~tbrun/Course/lecture11.pdf

[19] T. E. Oliphant, *A guide to NumPy*. Trelgol Publishing USA, 7 Dec. 2006.

Initial Distribution List

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California