The Journal of Supercomputing manuscript No.
(will be inserted by the editor)

Experience in Teaching Quantum Computing with Hands-on Programming Labs

Federico Galetto · Hiram H. López · Mehdi Rahmati · Janche Sang · Chansu Yu

Received: Dec. 18, 2023 / Accepted: date

Abstract As the field of quantum computing rapidly advances, there is a growing demand for skilled professionals adept in quantum computing and programming. Recognizing this need, in this paper, we share our experiences teaching an introductory-level quantum computing course to students at Cleveland State University (CSU). The course integrates dedicated hands-on programming labs, allowing students to verify their experimental results with corresponding examples from the textbook. These labs cover a diverse range of topics, including fundamental elements such as quantum gates and circuits, quantum key distribution protocols, and quantum algorithms. As educators, our goal is to share teaching insights and resources with fellow instructors in the field. This article elucidates the rationale behind the design of each experiment, providing a deeper understanding of quantum computing.

Keywords Quantum Parallelism · Quantum Computing Education · Hands-On Laboratory · Quantum Programming · Qiskit

1 Introduction

Quantum Computing is a rapidly growing field with many potential applications, including cryptography, optimization, pharmaceuticals, materials science, etc. As more

F. Galetto
Dept. of Mathematics and Statistics, Cleveland State University, Cleveland, OH, USA

H. H. López
Dept. of Mathematics, Virgina Tech, Blacksburg, VA, USA

M. Rahmati
Dept. of Electrical and Computing Engineering, Cleveland State University, Cleveland, OH, USA

J. Sang (Corresponding author)
Dept. of Computer Science, Cleveland State University, Cleveland, OH, USA

C. Yu
Dept. of Electrical and Computing Engineering, Cleveland State University, Cleveland, OH, USA

and more companies and organizations invest in quantum research and development, the demand for skilled professionals with expertise in quantum computing is likely to increase. That is, having an understanding of quantum computing and practical experiences in quantum programming can be a valuable asset for Computer Science or Computer Engineering students looking to pursue careers in industry or academia. Therefore, we offered an introductory-level Quantum Computing course to our senior students at CSU, for the first time in Fall 2022.

Recommended by Prof. Kiper at Miami University [16], we chose Bernhardt's book *"Quantum Computing for Everyone"* [1] as the textbook. This book provides simple explanations of the challenging mathematics of quantum computing and gives elementary examples to illustrate what it means and how it is applied in problem solving. However, like many other quantum computing books, this textbook also lacks practical quantum programming content, which would be helpful for EECS students because quantum programming expertise will likely be in high demand as quantum computing continues to advance.

Hands-on programming provides a means to bridge the gap between theoretical knowledge and practical implementation in quantum computing. By engaging in hands-on programming, students can gain a deeper understanding of quantum concepts such as superposition, entanglement, quantum gates and quantum algorithms. For example, it might be easier for students to understand how the Deutsch [7] and the Deutsch-Jozsa [8] algorithms work via the quantum phase kickback circuit implementation rather than the rigorous proof in math. Furthermore, by writing code and running it on quantum computers or simulators, students can explore the real-world implications of quantum algorithms and discover novel ways to leverage quantum computing power. As suggested in [19], a course with dedicated programming labs would greatly improve students' learning experience in quantum programming.

Several open-source quantum software development kits, such as Amazon's Braket, Google's Cirq, IBM's Qiskit, Microsoft's Q# and QDK, etc. are available on-line. A good survey and comparison of these frameworks can be found in [9]. For the dedicated hands-on labs, we focused on teaching students a single quantum programming environment throughout the course. We chose IBM's Qiskit because IBM is the only provider which allows unlimited free accesses to actual quantum computers. Furthermore, the Cleveland Clinic [4] installed the IBM Quantum System One [12], which supports a 127-qubit Eagle processor, to accelerate biomedical discoveries. Through the collaboration between CSU and the Cleveland Clinic [29], our students will have opportunities to conduct experiments on the cutting-edge quantum computer.

The objective of this paper is to narrate our teaching experiences in the field of quantum computing. Additionally, we aim to elucidate the design of hands-on experiments that empower students to execute Python-based quantum programs across seven dedicated lab sessions. Many of the experiments are tied to the textbook, and the students are asked to modify a program template and verify their experimental results with the corresponding example described in the textbook. To give students more practice, we selected and revised some experiments from the Qiskit Textbook [21], Qiskit Computing Labs [22], as well as from the book by Professors Yanofsky and Mannucci [30]. Through this article, we want to share our teaching experience and materials with other instructors who are teaching quantum computing. Interested readers can find the full handouts and a detailed explanation of each experiment in [14].

The organization of this paper is as follows. Section 2 describes the related work in quantum computing education. Section 3 presents the course content and our teaching experiences. In Section 4, the rationales behind the design of each experiment and some details of the experiments are presented. Section 5 presents the survey results. Finally, we give a short conclusion in Section 6.

2 Background and Related Work

The phenomenon of quantum parallelism stems from the ability of quantum qubits to exist simultaneously in multiple states through superposition. This capability empowers quantum computers to explore numerous possibilities in parallel, potentially solving complex problems exponentially faster than their classical counterparts. Figure 1 shows a general quantum programming model that encompasses both quantum and classical components. The quantum component comprises the quantum circuits constructed using qubits and quantum gates. Quantum gates, such as Hadamard gates and others, are employed to manipulate the quantum states of qubits, with Hadamard gates being particularly valuable for creating superposition states. As depicted in Figure 1, the Hadamard gates can concurrently generate all 2^n input cases for the quantum circuits. These circuits encode the quantum algorithm and can be repeated depending on the specified problem, as exemplified by Grover's Search algorithm [11].

The classical component serves as the controller of the quantum computation. After each quantum computation round, the classical component gathers the measurement results from the qubits. If deemed necessary, the classical component directs the quantum component to perform more rounds of quantum computation. This process can be repeated within a predefined time frame or until specific convergence criteria are met. Subsequently, the classical component aggregates all the measurement

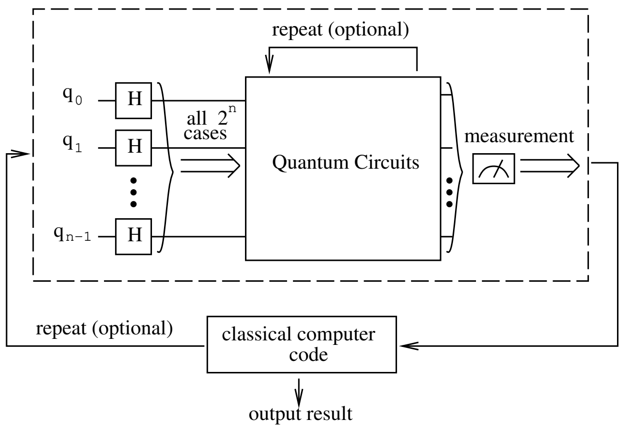


Fig. 1: A General Quantum Programming Model

results and processes them to obtain the final computation results, as illustrated by algorithms like Simon's algorithm [27] and Shor's algorithm [26].

Traditional quantum computing courses usually focus on physical and mathematical concepts and most of them target graduate students. Over the past three years, an increasing number of universities have been offering introductory quantum computing courses for undergraduate students, emphasizing the practical aspects of quantum computing. As described in the paper [5], the instructors gave online lectures worldwide and demonstrated running a set of Qiskit quantum programs with Jupyter notebooks. In [3], the authors compared various quantum programming environments and chose Qiskit as the framework for teaching. They instructed students in various examples, including quantum teleportation, the Deutsch–Jozsa algorithm, the Bernstein–Vazirani algorithm, Simon's algorithm, the Quantum Fourier transform, and others, all of which were supported by Qiskit Quantum Computing Labs web pages. A practical software-driven approach for teaching quantum computing was proposed in [19]. Their students needed to use Microsoft Quantum Katas to complete six programming assignments and submit a final programming project. Similarly, the teachers presented their Computer Science-oriented approach in [25] by using approximately 30 Jupyter notebooks, focusing on practical concepts and teaching Qiskit-based quantum programming in a three-day-long workshop.

The major difference between our approach and the aforementioned previous work is that we offered dedicated hands-on labs, providing students with in-person access to the instructor or lab TA for assistance when needed. Furthermore, many of the experiments are closely linked to the content of the textbook, and these experiments are designed to be progressive, with each new experiment building upon the concepts and skills learned from the previous ones. This approach allows students to gradually apply their knowledge, ensuring a deeper understanding as they progress through the course.

3 Course Content and Teaching Insights

We outline the course content and schedule in Table 1. The class, scheduled weekly for 3 consecutive hours, includes a 75-minute hands-on lab section immediately after discussing the topic. Performing a lab after the topic ensures that the information remains fresh in students' minds. During a hands-on lab, students can observe the results of their actions in real time. This instant feedback allows students to identify and correct misconceptions or errors, further reinforcing their learning and understanding of the material.

Over the past two years, five professors actively contributed to the preparation and teaching of this course. Dr. Yu took the lead as the course initiator and coordinator. He was also responsible for inviting the guest speaker and delivering the general introduction during the first-week lecture. Dr. López and Dr. Galetto, affiliated with the Department of Mathematics and Statistics, instructed on linear algebra, quantum spin, and qubits in Fall 2022 and Fall 2023, respectively. Dr. Rahmati dedicated his efforts to teaching quantum gates, entanglement, Bell's inequality, superdense code,

Table 1: Topics and Schedule

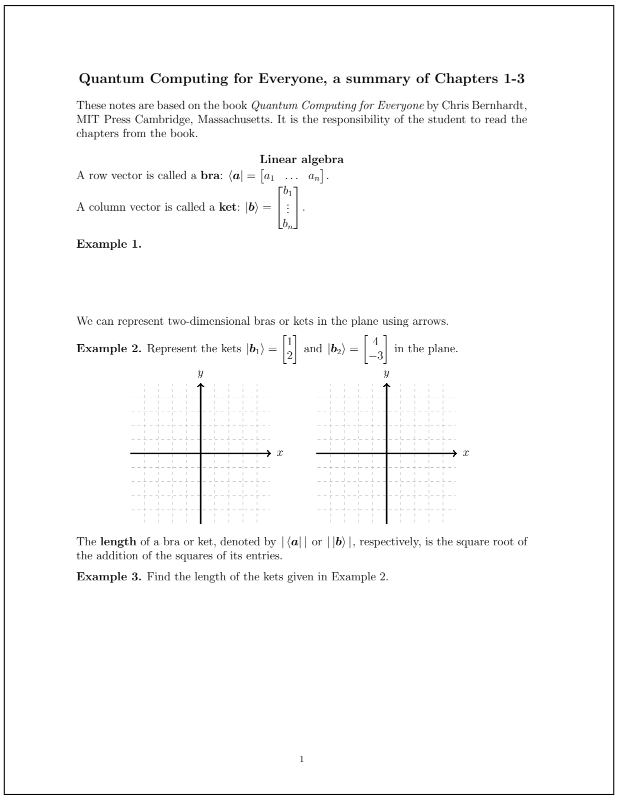
	Topic 1	Topic 2
Week 1	Introduction (Ch. 1)	Overview: Qubit, Superposition, Interference & Entanglement
Week 2	Linear Algebra (Ch. 2)	Linear Algebra (Ch. 2)
Week 3	Spin & Qubits (Ch. 3)	Lab 1 (Python & Jupyter)
Week 4	Spin & Qubits (Ch. 3)	Quantum Gates (Ch. 7)
Week 5	Test 1	Entanglement (Ch. 4)
Week 6	Entanglement (Ch. 4)	Lab 2 (Qiskit and Entanglement)
Week 7	Bell's inequality (Ch. 5)	Lab 3 (Quantum Gates and Circuits)
Week 8	Ekert, QKD (Ch. 5), Lab 4 (QKD)	Guest Speaker (Cleveland Clinic)
Week 9	Test 2	Superdense code, Teleportation & Error Correction (Ch. 7)
Week 10	Deutsch & Deutsch-Jozsa algorithms (Ch. 8)	Lab 5 (Deutsch algorithm)
Week 12	Simon's algorithm (Ch. 8)	Lab 6 (Simon's algorithm)
Week 14	Shor's algorithm & Grover's algorithm (Ch. 9)	Lab 7 (Grover's algorithm and 3-SAT)
Week 15	Complexity (Ch. 8)	Test 3

teleportation and error correction. Meanwhile, Dr. Sang led the hands-on lab sessions and instructed on quantum algorithms.

Dr. López wrote the first version of the skeleton notes in LaTeX for Fall 2022, and Dr. Galetto made minor revisions for Fall 2023. Before starting the class, we gave every student a set of print skeleton notes. Figure 2 shows an example of one page from these partial notes. During class, we filled out the gaps in the notes on a tablet device connected to a projector while students did the same on their print set. We uploaded the filled notes to our LMS for students to peruse right after class. The pedagogical value of the skeleton notes (also known as partial notes) has been documented in the literature [2].

We begin by reviewing the necessary mathematical tools to represent qubits and use them in quantum computing. Following [1], we represent qubits with unit vectors in the real plane. Thus, we must review some facts about vectors (addition, scalar multiplication, dot product, length, and orthogonal bases) and matrices (multiplication, transpose, and orthogonal matrices). All the necessary notions are covered in the prerequisite linear algebra course (MTH 288 at CSU). As the students may take the quantum computing class anywhere from one to two years (three for graduate students) after completing linear algebra, we prefer to review the mathematical background as part of the course.

A common question is the difference between bras and kets, represented respectively by row and column vectors. From a mathematical perspective, bras are elements of the vector space dual to the one containing the kets; however, this point of view is too sophisticated. Fortunately, physics comes to the rescue to offer a more tangible description. To each position of the Stern-Gerlach apparatus, we associate an orthonormal basis of the plane, say $\{|\mathbf{b}_0\rangle, |\mathbf{b}_1\rangle\}$, whose elements correspond to the two possible states of an electron that passed through the apparatus. Let $|\mathbf{b}\rangle$ be a



Quantum Computing for Everyone, a summary of Chapters 1-3

These notes are based on the book *Quantum Computing for Everyone* by Chris Bernhardt, MIT Press Cambridge, Massachusetts. It is the responsibility of the student to read the chapters from the book.

Linear algebra

A row vector is called a **bra**: $\langle \boldsymbol{a}| = \begin{bmatrix} a_1 & \dots & a_n \end{bmatrix}$.

A column vector is called a **ket**: $|\boldsymbol{b}\rangle = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix}$.

Example 1.

We can represent two-dimensional bras or kets in the plane using arrows.

Example 2. Represent the kets $|\boldsymbol{b}_1\rangle = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$ and $|\boldsymbol{b}_2\rangle = \begin{bmatrix} 4 \\ -3 \end{bmatrix}$ in the plane.

The **length** of a bra or ket, denoted by $|\langle \boldsymbol{a}||$ or $||\boldsymbol{b}\rangle|$, respectively, is the square root of the addition of the squares of its entries.

Example 3. Find the length of the kets given in Example 2.

1

Fig. 2: Partial class notes about Linear Algebra

unit vector representing a qubit. Then, we have

$$|\mathbf{b}\rangle = \langle \mathbf{b}_0|\mathbf{b}\rangle\, |\mathbf{b}_0\rangle + \langle \mathbf{b}_1|\mathbf{b}\rangle\, |\mathbf{b}_1\rangle\,,$$

where the square of the braket (dot product) $\langle \mathbf{b}_i|\mathbf{b}\rangle$ is the probability that the qubit will be found in the state $|\mathbf{b}_i\rangle$ after passing through the apparatus. Thus, unit kets represent qubits in a certain state, while unit bras are operators that measure properties of qubits.

We developed homework assignments and assessments with examples similar to the ones covered in the class. The main goal is that students excel in the math concepts for the rest of the course. For instance, a vector representation in terms of an orthonormal basis should be for students as familiar as multiplying two matrices. For the future, we plan to create a similar version of the homework problems in the online platform MyOpenMath, to provide students with more practice opportunities and instant feedback.

It is worth mentioning that we provided each student with three 2”×2” polarized sheets to conduct the interesting experiment described in the textbook. Note that qubits are typically represented by the spin of electrons or the polarization of photons. An essential feature of qubits can be directly observed through experimentation using polaroids to filter photons based on polarization. Firstly, incoming light is randomly polarized. Using a polaroid (aligned horizontally), approximately half of the photons pass through, resulting in an output intensity of half. Subsequently, the outgoing photons are all horizontally polarized. Secondly, introducing a second polaroid (rotated 90 degrees, polarized vertically) prevents any photons from passing through since they are horizontally polarized. Consequently, the output intensity becomes zero (complete darkness). Thirdly, inserting a third polaroid between the two existing ones (rotated 45 degrees) leads to a counterintuitive observation: the additional filter increases the output intensity. Some photons manage to pass through all three polaroids, as shown in Figure 3. This experiment illustrates a key aspect of quantum mechanics, emphasizing that the outcome of a measurement can vary based on the chosen basis (in this case, polarization). Different bases may provide distinct information about the quantum state, highlighting the inherently probabilistic nature of quantum systems.

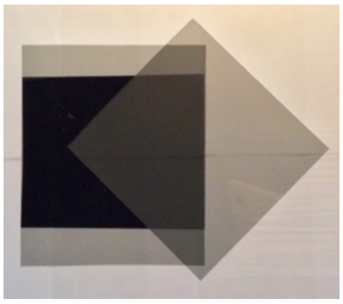


Fig. 3: Overlapping Three Linear Polarized Sheets

Dr. Rahmati kicked off his section by talking about quantum gates and circuits. At first, students had some questions to understand the functionality of these quantum gates, especially those students who are used to working with classical logic gates. However, with examples and discussions, we made it easier to see how quantum and classical computing are different. We spent quite a bit of time in class talking about the no-cloning theorem, proving it, and showing why it does not match our traditional way of thinking about regular computers. One particularly challenging concept, i.e., entanglement, was explained using the tensor product and by providing a clear comparison of entangled and unentangled qubits. The CNOT gate, as a visual tool, aided students' grasp of entanglement.

Our learning journey delved into the history of quantum mechanics, connecting iconic figures like Einstein and Bohr and explaining what happened during that time. The students liked it very much. We continued the discussion by clearly explaining Bell's inequality and then showed how it can be used in quantum key distribution. This is followed by explaining superdense coding, teleportation, and an introductory lecture on quantum error correction and the need for new algorithms to apply to existing quantum computers. Some questions arose about the potential adaptation of certain classical error correction algorithms for use in quantum error mitigation and correction, which was really inspiring to see students engaged with the topics.

Lastly, Dr. Sang covered fundamental quantum algorithms such as the Deutsch and Deutsch-Jozsa algorithms, Simon's periodicity algorithm, and Grover's search algorithm. Our approach followed the structure laid out in the textbook, ensuring a comprehensive understanding of these quantum computing concepts. Specifically, for the Deutsch and Deutsch-Jozsa algorithms, we enhanced the learning experience by incorporating circuit visualization alongside mathematical derivations. This dual approach aids students in grasping both the theoretical foundation and practical aspects of these algorithms. In the homework assignment, we introduced variations by altering the initial value of the x qubit from 0 to 1, prompting students to derive the mathematical solutions and further reinforcing their comprehension through circuit visualization.

Shifting our focus to quantum complexity classes, our curriculum explores Quantum Polynomial Time (QP) and Bounded-error Quantum Polynomial Time (BQP). We delved into the theoretical aspects of quantum computing, discussing its potential impact on classical cryptography and computational complexity. Notably, we addressed the vulnerability of RSA encryption through Shor's algorithm and the potential speedup for NP-Complete problem solving via Grover's search algorithm. The exploration of algorithms and complexity classes provides students with a comprehensive understanding of quantum computing's theoretical underpinnings and practical implications.

4 Hands-on Quantum Programming Labs

The seven dedicated hands-on labs were conducted in our Linux cluster lab. Each workstation has installed the software packages and libraries, such as Python3, qiskit, jupyter, etc. Note that the Jupyter Notebook [28] is an interactive computing environment and its cell-based approach makes it easy to organize code and text, run code interactively, and create documents that combine executable code, visualizations, and explanations.

In the lab, students need to follow the procedure stated in the handout step by step to implement the programs, record, and explain the results in their lab reports. This section gives a brief description of each lab and its objectives. Some interesting experiments which are important for students to practice are also presented below. Interested readers can find the full handouts and the detailed explanation of each experiment in [14].

4.1 Hands-on Lab 1: Python and Jupyter

In the first warm-up lab, each student will set up their working environment (e.g. installing Qiskit tools, creating a dedicated sub-directory for this course, etc.), and get familiar with some Linux commands. Students will also learn how to implement and run Python 3 programs in Jupyter Notebook's code cells, as well as how to load and execute program templates in cells. Furthermore, they will review some features in the Python language, such as using indentation instead of a pair of curly braces in C/C++/Java to indicate a block of code, the dictionary data structure which uses string as the array index, etc. Note that, in this lab, students are asked to copy and run the quantum ”Hello World” program which uses a Hadamard gate to set a qubit into a superposition state and then measures it. However, we will not explain how the code works in detail until the next lab. This fosters a sense of curiosity that may lead to more profound and lasting learning outcomes.

4.2 Hands-on Lab 2: Qiskit and Entanglement

In this lab, students will learn how to write simple quantum programs in Qiskit. A Qiskit program typically follows a structured pattern to firstly construct the quantum circuits using the Qiskit Terra module and then execute the circuits either on a simulator (i.e. the Qiskit Aer module) or on a real IBM quantum computer. Instructors should remind students that the Qiskit’s qubits order is simply reversed as compared to the order used in the textbook [1] (also in the reference [30]). That is, Qiskit puts q_0 rightmost (e.g. $q_2q_1q_0$) while the textbook puts q_0 leftmost (e.g. $q_0q_1q_2$). Students have to be careful while reading Qiskit documents, Qiskit gate operation matrix, histogram output result, etc.

Students will study in detail the quantum ”Hello World” program template given in the previous lab. Next, they will learn that quantum randomness via the H gate is not simply like a classical random coin toss. That is, as shown in Figure 4, students will modify the template by applying two H gates in succession and observe that the result is not exactly the same as tossing the coin twice. They also need to verify the result through the state vector calculation.

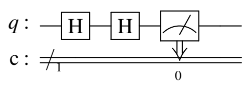


Fig. 4: Applying two H gates in succession

The major theme of Lab 2 is entanglement. Students will construct the two-qubit entanglement circuit shown in Figure 5 and observe the entangled results (i.e. either ”00” or ”11”). In the next experiment, they need to modify the circuit by initializing the qubit q_1 to be 1 like in the Figure 6. Students are asked to explain whether the result is entangled or not in their reports. They can refer to page 59 in the textbook about the simple rule to check the entanglement of two qubits. That is, assume that r, s, t, and u represent the probability amplitudes of two qubits q_0q_1 states: 00, 01, 10, and 11, respectively. These two qubits are entangled if $ru \neq st$. Otherwise, they are not entangled and can be decomposed into the tensor product of two individual qubits.

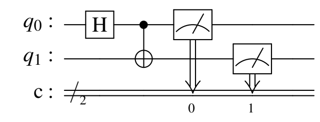


Fig. 5: Two-qubit Entanglement: 00 or 11

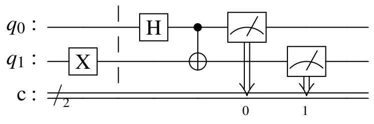


Fig. 6: Two-qubit Entanglement: 01 or 10

After finishing Lab 2, students will be given a homework assignment. They are asked to run a two-qubit entanglement program on an actual quantum computer. Each student needs to follow the steps to create a free account on the IBM website, which allows the user to submit and execute quantum programs on IBM quantum computers worldwide [23]. They need to explain in their report why the histogram result is a little different from the result obtained in the Lab2 Experiment using the simulator.

4.3 Hands-on Lab 3: Quantum Gates and Circuits

This lab starts with doing two experiments: one is to entangle three qubits from the same control qubit and the other is to entangle four qubits in a chain manner. Next, students are asked to use quantum gates such as X, H, CNOT, and Toffoli, to construct circuits for emulating the classical AND, NAND, and OR gates. Some students may have learned NP-completeness already. Instructors could tell them that, for NP-complete problems, such as SAT, quantum computers can give quadratic speedup over the classical computers. Learning how to emulate the classical gates will help them build the oracle circuits used in the Grover's search algorithm [11] for solving the SAT problem. These SAT-related experiments will be conducted in Lab 7 in the future.

To emulate the classical AND gate function, we can simply use a Toffoli gate `ccx(c0,c1,t)`, where `c0` and `c1` are the control qubits and `t` is the target qubit. When both `c0` and `c1` are set to 1, the target qubit `t` will be flipped to 1. To implement the NAND gate function, we just need to add a NOT gate (i.e. the quantum X gate) to flip the target qubit `t` of the Toffoli gate at the end.

Note that there are two different methods to implement the classical OR gate. One is to use two CNOT gates and one Toffoli gate, as shown in Figure 7. If either one of the two inputs is 1, it uses the CNOT to toggle the output result to be 1. If both inputs are 1's, output result will be toggled twice back to 0 and hence we need the Toffoli gate to toggle the output qubit again. The other method is to adopt X and AND gates based on the following equation:

$$q_0 \vee q_1 = \neg(\neg q_0 \wedge \neg q_1)$$

That is, as shown in Figure 8, we invert both inputs and apply the NAND function. Finally, we have to invert the inputs back to their original values. It seems that there is no significant difference between these two methods since both of their circuit depths, which calculates the longest path between the data input and the output, look like 3. However, it has been shown that it needs at least five two-qubit gates to implement the Tofolli gate [31]. Therefore, the first approach has a larger circuit depth because it uses two more Tofolli gates than the second approach.

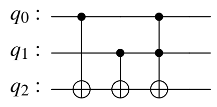


Fig. 7: Implementing the two-qubit OR function with CNOT and Toffoli gates

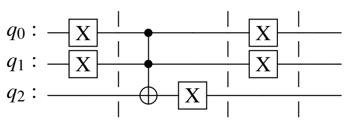


Fig. 8: An alternative implementation of the two-qubit OR function

There are two slightly more complicated quantum circuits that are useful for students to study. The first circuit is shown in Figure 9. After implementing and executing the program, students will find out, for the four possible inputs q_0q_1: 00, 01, 10, and 11, the corresponding outputs will be: 00, 10, 01, and 11, respectively. They need to deduce that the circuit swaps the states between q_0 and q_1.

The second circuit, known as Phase Kickback and depicted on the upper left of Figure 10, will generate the outputs q_0q_1 as 00, 11, 10, and 01, corresponding to the inputs: 00, 01, 10, and 11, respectively. Namely, the circuit acts like the q_1 is the control while the q_0 becomes the target of a CNOT gate, as shown in the upper right of Figure 10. Note that the two H gates in q_0's wire are not successive due to the presence of a CNOT connection in between. Students are recommended to use matrix operations, as shown in the bottom part of Figure 10, to verify the circuit

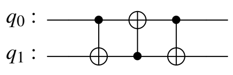
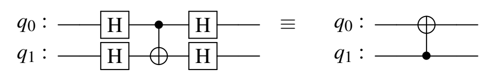


Fig. 9: Swap the states in q_0 and q_1



$$\frac{1}{2}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \times \frac{1}{2}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \equiv \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

Fig. 10: Phase Kickback and Corresponding Matrix Operations

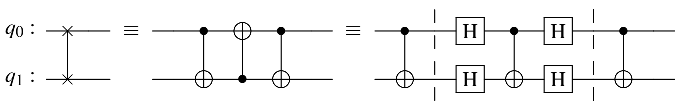


Fig. 11: Swap the states in q_0 and q_1 if q_1 cannot be the control qubit

equivalence. The Phase Kickback circuit has been used in many quantum algorithms. In our experience, comprehending the Phase Kickback circuit would be helpful for beginners in elucidating the workings of the Deutsch and Deutsch-Jozsa algorithms.

It is also worth mentioning that, though not included in the Lab3, the circuits in Figure 9 and Figure 10 can be used together. The current IBM quantum computer adopts the heavy-hexagon lattice topology to connect qubits [20]. Hence, the swap gate is needed to facilitate interactions between non-adjacent (i.e. not directly connected) qubits. Moreover, the control qubits are not adjacent in the heavy-hexagon lattice and this limitation makes the circuit in Figure 9 impossible if q_0 and q_1 are directly connected together. If the qubit q_1 in Figure 9 cannot be the control qubit, we can simply use the Phase Kickback to solve the problem [32]. The solution is depicted in the rightmost part in Figure 11. This can be left as a homework exercise for students.

In addition to the swap gate, there is another interesting quantum circuit, called the bridge gate [13], which lets the CNOT gate operate on non-adjacent qubits. For example, if the control qubit q_0 and the target qubit q_2 are not adjacent, they can use their common neighbor q_1 as the bridge. This involves four direct-connect CNOT gates, as shown in Figure 12. The circuit can be comprehended using the following instructions in sequence:

$$q_2 = q_{1.org} \oplus q_{2.org}$$
$$q_1 = q_{0.org} \oplus q_{1.org}$$
$$q_2 = q_1 \oplus q_2 = (q_{0.org} \oplus q_{1.org}) \oplus (q_{1.org} \oplus q_{2.org}) = q_{0.org} \oplus q_{2.org}$$
$$q_1 = q_0 \oplus q_1 = q_{0.org} \oplus (q_{0.org} \oplus q_{1.org}) = q_{1.org}$$

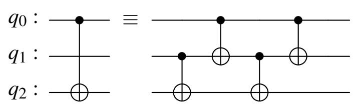


Fig. 12: The Bridge gate: The CNOT gate (control: q_0, target: q_2) by way of q_1

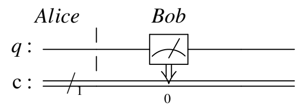


Fig. 13: BB84 emulation: Both Alice and Bob choose the Vertical direction

Note that the notation q_{org} denotes the original value of the qubit q. It can be seen that the value in q_2, the target qubit of the CNOT gate, achieves its desired result, while the intermediate qubit q_1 remains unchanged at the end. We plan to include this example as a new experiment in the future.

4.4 Hands-on Lab 4: Quantum Key Distribution

Quantum Key Distribution (QKD) is a cryptographic technique that utilizes quantum mechanics to enable two communication parties to produce a shared random secret key known only to them. The secret key can be used for encrypting and decrypting messages. There are two QKD protocols discussed in the textbook: BB84 and Ekert91. In this lab, students are asked to implement Qiskit programs to emulate the behaviors of these two protocols and verify the output results of the probabilities from their programs with the corresponding descriptions in the textbook.

The BB84 protocol exploits the principles of quantum mechanics, utilizing the properties of quantum states to ensure the security of the key exchange process. As described in the textbook, Alice prepares a sequence of $4n$ qubits, encodes each qubit randomly in one of two bases (i.e. either vertical or horizontal) and then sends them to Bob. Bob measures each of these qubits using randomly chosen bases. Then, they publicly communicate which bases they used for each qubit and only keep the qubits corresponding to the times when they both use the same basis. The length of these kept qubits is roughly about $2n$. These qubits will be identical if Eve is not intercepting them. Therefore, Alice and Bob can compare half of these qubits to detect the presence of any Eve's eavesdropping attempts. If they agree on all of them, they know Eve is not listening in and they can use the other n qubits as the shared key to encrypt/decrypt messages in the future communications.

The first simple experiment is to let students observe that Alice and Bob will obtain the same qubit value if they choose the same basis – either the vertical or horizontal direction, as shown in Figure 13 and Figure 14, respectively. In the next

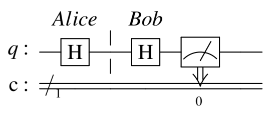


Fig. 14: BB84 emulation: Both Alice and Bob choose the Horizontal direction

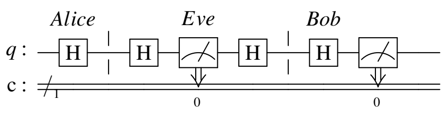


Fig. 15: BB84 emulation with Eve's eavesdropping and guessing right

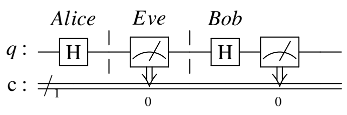


Fig. 16: BB84 emulation with Eve's eavesdropping and guessing wrong

experiment, students are asked to run a template program which emulates the BB84 protocol except detecting any interception. The template initially sets the *n* to be 4, a small value so students can trace the executions easily. Based on the program output, they need to mark, in their report, the bit positions where Alice and Bob use the same bases and then extract the corresponding bits from ”Alice bits” and ”Bob results” into two bit sequences. Finally, they need to add the code to select half of the same-basis bits and compare them to determine whether they are the same or not. If the bits comparison is True, students can increase the variable *n* to a larger value, say 1000, and run the program again to make sure that their program works for a larger size of bit sequence.

In the third experiment, students are asked to add lines of code to their program implemented in the first experiment to emulate Eve's interception. It can be observed that if Alice, Eve, and Bob choose the same basis, they will all get the same bit and hence Eve's eavesdropping cannot be detected. Figure 15 is an example in which Alice, Eve, and Bob all choose the horizontal basis. Note that after Eve guesses the measurement basis, she will send the qubit to Bob using the same basis. If Eve chooses the wrong basis, as shown in Figure 16, the emulated program will output that Bob gets the right bit with only a 50% chance.

In Experiment 4, students need to modify the code implemented in Experiment 2 to emulate Eve's actions. That is, Eve firstly generates a random choice of basis for each bit. Next, she measures each qubit based on the generated random choice of basis. Finally, she encodes the message again based on the generated random choice

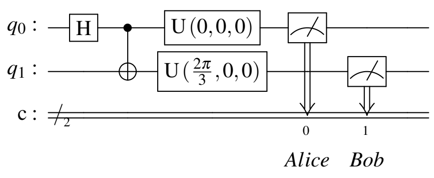


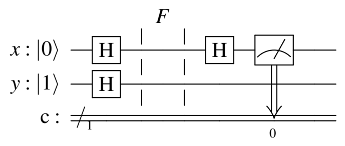
Fig. 17: Ekerti91 emulation: Measuring two entangled qubits with different bases

of basis and then sends the message to Bob. Students are also asked to add some code to find out the percentage of the n sample bits in which Alice and Bob disagree. As described in the textbook, if Alice and Bob disagree about a quarter of the sample bits (i.e. $n/4$), they know that Eve's intercepting. Students can check their program's output whether it is $1/4$ or not.

Experiments 5, 6, and 7 are about the Ekert91 protocol which utilizes entangled qubits as in the Bell's test. There are a few variations of the protocol. Based on the version presented in the textbook, there are $3n$ pairs of entangled qubits. For each pair, Alice receives one and randomly chooses a direction to measure her qubit, while Bob gets the other and also randomly selects a direction to measure it. There are three directions Alice and Bob can use: $0°$, $120°$, or $240°$. Therefore, they will agree around n bits when both choose the same direction. Because the bases they chose will be revealed over a public insecure line, they can use these same-basis n bits as the shared secure key if Eve is not intercepting. Note that to detect interception, they use the sequence of qubits that come from the times when they chose different bases.

The template program provided in Experiment 5 simply tests one pair of entangled qubits when Alice and Bob measure the qubits using different directions via the Qiskit U-gate (see Figure 17). From the program output, students need to calculate the probability both Alice and Bob's qubits agree (i.e. combining the percentages of '00' and '11') and the probability they disagree (i.e. adding the percentages of '10' and '01'). In addition to $0°$ and $120°$ different directions, students also need to test the other five cases such as $0°$ and $240°$, $120°$ and $240°$, etc. Students will find out that for any case, the probability Alice and Bob agree is approximate 0.25 and the probability they disagree is around 0.75. The theoretic proof can be found in the section about Bell's inequality in the textbook.

In Experiment 6, students need to study, modify and run another template program which emulates Alice's and Bob's behaviors based on the Ekert protocol. This program outputs the probability that Alice and Bob agree when they use different bases. In the next experiment, they are asked to modify and extend this template program to detect whether Eve is intercepting or not. If Eve is eavesdropping, the probability that Alice and Bob agree when they use different bases will raise from 1/4 as in the Experiment 6 to 3/8. Students can check whether their results are consistent with the description in the textbook. Note that the textbook does not derive how to get the probability 3/8 in detail. This can be left as an homework assignment for students.



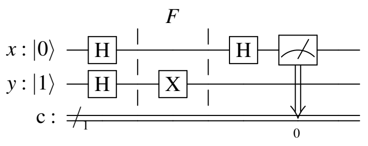


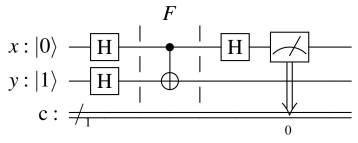
Fig. 18: Deutsch — the Constant 0 function (left) and the Constant 1 function (right)

4.5 Hands-on Lab 5: Deutsch and Deutsch-Jozsa Algorithms

The Deutsch algorithm is the first algorithm to show that a quantum algorithm could be faster than a classic one to solve a specific problem. It specifically addresses the task of determining whether a black-box function $f : \{0,1\} \longrightarrow \{0,1\}$ is constant or balanced. Note that a function is called balanced if it sends half of its inputs to 0 and the other half to 1. There are four such functions. Two are constant: the constant 0 function ($f(0) = f(1) = 0$), or the constant 1 function ($f(0) = f(1) = 1$), while the other two are balanced: the identity function ($f(0) = 0$, $f(1) = 1$), or the inversion function ($f(0) = 1$, $f(1) = 0$). In classical computing, it would typically require two function evaluations to tell whether a black-box function is constant or balanced. By employing quantum principles such as superposition and interference, the quantum algorithm requires only a single query to the function, providing a speedup over the classical method.

In the first four experiments of this lab, students will learn how to construct the circuits for each of the four black-box functions. For a quantum system, every operation must be unitary and thus reversible. For achieving this, each function, say F, needs another input y to map the state $|x,y\rangle$ to the state $|x,y \oplus f(x)\rangle$. Because $|x,(y \oplus f(x)) \oplus f(x)\rangle = |x,y \oplus 0\rangle = |x,y\rangle$, the function F is its own inverse. To construct the circuit for the constant 0 function, where $f(x) = 0$, the output of the bottom qubit becomes $y \oplus f(x) = y \oplus 0 = y$. Namely, there are only two straight wires in the black box F and there is no connection between the qubit x and the qubit y, as shown in the left part of Figure 18. Hence, the top qubit x goes through two H gates in succession. Based on the experience we learned from Experiment 2 in Lab 2, the qubit x will be restored back to its initial value 0 before measurement. Similarly, the circuit for the constant 1 function, where $f(x) = 1$, can be constructed as in the right part of Figure 18. This is because $y \oplus f(x) = y \oplus 1 = \bar{y}$. Same as the constant 0 function, there is no connection between the qubit x and the qubit y. The qubit x also goes through two successive H gates and its value will be restored back to its initial value 0. Therefore, both constant functions yield the value 0 in the qubit x.

The left portion of Figure 19 shows how to build the circuit of the balanced identity function, where $f(x) = x$. That is, the function F adopts a CNOT gate to map the input state $|x,y\rangle$ to the output state $|x,y \oplus f(x)\rangle = |x,y \oplus x\rangle$. From the Experiment 7 in the Lab 3 (i.e. the phase kickback circuit), we know that bottom qubit y becomes the control bit of the CNOT gate. Furthermore, another tricky setting is the initial value of the qubit y. The value 1 in y will let the qubit y toggle the value in the qubit x from 0 to 1, which is different than the result of the constant functions. Likewise,



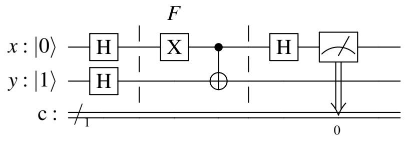


Fig. 19: Deutsch — the balanced functions; either Identity (left) or Inversion (right)

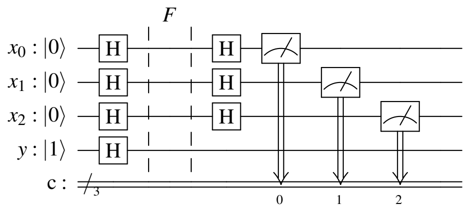


Fig. 20: Deutsch-Jozsa – the constant 0 function $f(x_0,x_1,x_2) = 0$

the circuit of the balanced inversion function, where $f(x) = \bar{x}$, can be constructed as in the right portion of Figure 19. The function F maps the state from $|x,y\rangle$ to $|x,y \oplus f(x)\rangle = |x,y \oplus \bar{x}\rangle$. To implement it, we need to apply an X gate for the qubit x to invert its value. However, this X gate will not change the superposition state $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ in x because $|0\rangle$ becomes $|1\rangle$ and $|1\rangle$ becomes $|0\rangle$. Same as the balanced identity function, the value 1 in y will flip the value in the qubit x from 0 to 1. Hence, if we measure the qubit x and its value is 1, we know that the function is balanced.

In the class lecture, students learned that the Deutsch algorithm doesn't truly demonstrate the power of quantum parallelism because it employs a single query to distinguish between two cases: constant or balanced. The Deutsch-Jozsa algorithm generalizes the Deutsch problem with a larger domain. It determines whether a black-box function $f : \{0,1\}^n \longrightarrow \{0,1\}$ is constant (i.e. it always outputs the same value for all 2^n inputs) or balanced (i.e. half of the inputs go to 0 and the other half go to 1). The Deutsch-Jozsa algorithm shows the power of quantum parallelism by using only one query to the oracle, as compared to the worst case scenario $2^{n-1}+1$ function evaluations needed in the classical method. In Experiment 5, students are asked to load and run a template program involving a constant 0 function with $n = 3$, as depicted in Figure 20. Next, they are instructed to add an X gate on the qubit y to make the black box F a constant 1 function. The instructor would remind students that there is no connection between the x qubits and the qubit y for both constant functions. Hence, just like the constant functions in the Deutsch's algorithm, these three qubits $x_0x_1x_2$ go through two successive H gates and the value in $x_0x_1x_2$ will be restored back to "000".

In Experiment 6 and Experiment 7, students are asked to construct the circuits of the balanced functions $f(x_0,x_1,x_2) = x_0 \oplus x_1 \oplus x_2$ and $f(x_0,x_1,x_2) = x_0x_1 \oplus x_2$, respectively. As shown in Figure 21 and Figure 22, there is at least one CNOT connection between the x qubits and the qubit y. Since the phase kickback causes the

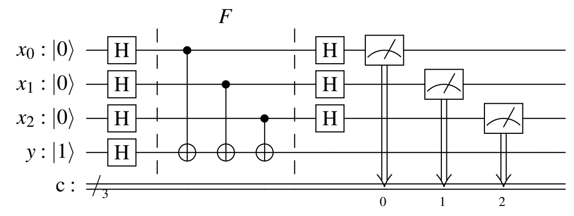


Fig. 21: Deutsch-Jozsa – the balanced function $f(x_0,x_1,x_2) = x_0 \oplus x_1 \oplus x_2$

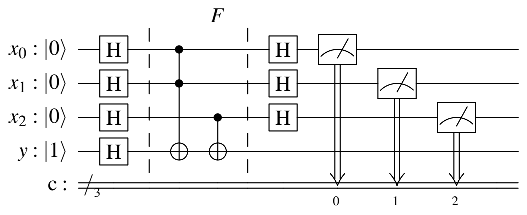


Fig. 22: Deutsch-Jozsa – the balanced function $f(x_0,x_1,x_2) = x_0 x_1 \oplus x_2$

input x qubits to be flipped through the connection(s), the value in $x_0x_1x_2$ will not be "000" anymore. That's why we can determine the function is constant or balanced by examining the x qubits.

We believe that these hands-on programming experiments will help students easily understand the Deutsch and the Deutsch-Jozsa algorithms from a different perspective, rather than relying solely on the rigorous mathematical proofs presented in the textbook. The phase kickback circuit, along with the y qubit's initial value set to 1, will flip certain values of the x-qubit for the balanced functions. This flipping will not occur for the constant functions due to the lack of any connection. However, while the mathematical derivation might be lengthy, it clearly demonstrates the relationship between the top x-qubit(s) and the bottom y qubit. As emphasized twice in the textbook, once for the Deutsch algorithm and once for the Deutsch-Jozsa algorithms, the x-qubit(s) and the y qubit are not entangled because their joint state can be expressed as a tensor product decomposition.

4.6 Hands-on Lab 6: Simon's Algorithm

Simon's algorithm solves the problem of finding the hidden string in a black-box function $f : \{0,1\}^n \longrightarrow \{0,1\}^n$. The black-box 2-to-1 function has the property that there is a secret non-zero binary string s of length n, (i.e.$s \neq 00\ldots0$), s.t. $f(x) = f(y)$ if and only if $y = x$ or $y = x \oplus s$. Simon's algorithm contains quantum procedures as well as classical post-processing procedures. As described in the textbook, we can run Simon's circuit $n+N$ times, where N does not depend on n, to get $n+N$

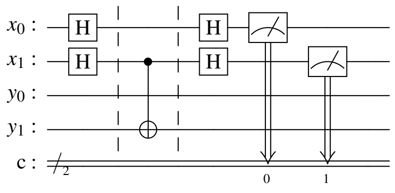


Fig. 23: A Simple Example of Simon's Algorithm Circuit with $s = 10$

equations. These equations contain the $n-1$ independent vectors, so we can use a classical algorithm (e.g. Gaussian elimination) to recover the secret string s.

Students begin this lab by running a 2-qubit template program with the hidden string $s = 10$. Figure 23 depicts the circuit of this program. Students should be reminded that, similar to the Deutsch-Jozsa algorithm, the top qubits x will be measured instead of the bottom qubits. This template outputs the top x qubits either 00 or 01. Note that the template does not call Gaussian elimination to find s and hence students need to manually solve the $n-1$ linearly independent equations. This will help them know how the whole algorithm works. Because the output string '00' which wouldn't give us any information, students need to use the other output string 01 to calculate its dot product with the secret string s (i.e. s_0s_1) and the result should be 0. That is, they will get

$$0 \times s_0 + 1 \times s_1 = 0.$$

They have to figure out that $s_1 = 0$ and hence $s = 10$ because not all of the digits in s can be 0. In fact, this example is from the textbook and has been discussed in the class lecture. Students now have the opportunity to practice it once again. In Experiment 2 and Experiment 3, students will modify the oracle circuit in the program based on the given diagrams in the handout. They may need to run their programs a few times to get $n-1$ different non-zero strings and then use these strings to derive the bit pattern of the secret string s.

In Experiment 4 and Experiment 5, students will learn a simple method, as presented in [24], for constructing an oracle circuit F using a given string s. Figure 24 shows the steps to construct such a 2-to-1 mapping function. Note that in the first step, because each qubit in the register y is initialized to be 0, the content of each qubit in the register x, which is the classical information encoded as either a $|0\rangle$ or a $|1\rangle$, will be copied through the CNOT gate to the corresponding qubit in the register y . In the second step, if $x_k == 0$, the register y remains unchanged. Otherwise, the register y will be XORed with s, i.e. $y \longleftarrow y \oplus s$. Figure 25 illustrates an example of using these two steps to construct the Simon's Algorithm Oracle Circuit with $s = 110$. Note that in this example, k is 0 because $s_0 = 1$. If the secret string s is 011, k is 1 since $s_1 = 1$ and hence the CNOT control bit is x_1 in the Step 2.

We need to show that the oracle F constructed using the method in Figure 24 has the property $f(x) = f(x \oplus s)$. Assume that the value in the x register is b when entering F. If b_k is 0, the register y has the value b when leaving F. Otherwise, it has the value $b \oplus s$. Consider another input value d, where $d = b \oplus s$, in the x register

1. Apply the CNOT gates from qubits of the first register (i.e. x) to qubits of the second register (i.e. y).
2. Because the bits in the string s cannot be all 0, find the least index k such that $s_k = 1$. Next, apply the CNOT gates from the qubit x_k to any qubit y_i if $s_i = 1$.

Fig. 24: A Simon's Oracle Construction Algorithm which implements $f(x) =$ if $x_k == 1$ return $(x \oplus s)$ else return x

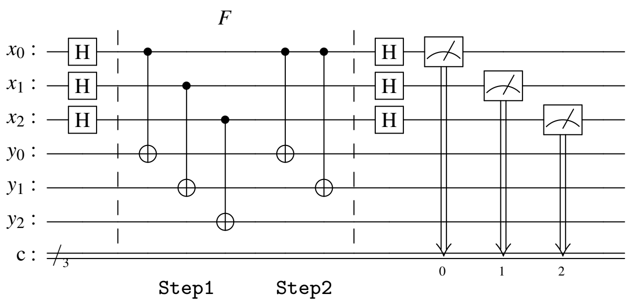


Fig. 25: An Example of Constructing Simon's Algorithm Oracle Circuit with $s = 110$

when entering F. Note that s_k is 1, so $d_k = b_k \oplus s_k = \overline{b_k}$. There are two cases we need to verify. When $b_k = 0$, $d_k = \overline{b_k} = 1$ and hence the register y will have the value $d \oplus s$ = $(b \oplus s) \oplus s = b$. If $b_k = 1$, then d_k is 0 and the register y will the value d which is $b \oplus s$. Therefore, for both cases, $f(b) = f(d) = f(b \oplus s)$.

There is another way to explain why the measured outputs of the top qubits are limited to certain patterns. Here, we employ the circuit equivalence translation along with the phase kickback property. We use the circuit in Figure 25 as an example. Firstly, we can simplify the circuit by removing the two successive CNOT gates from x_0 to y_0 because CNOT is its own inverse. Next, we add two successive H gates in several places to yield the circuit as shown in Figure 26. Adding two successive H gates does not change the circuit's property. Now, we can use the the phase kickback circuit pattern, as shown in in Figure 10, to reverse the control-target positions in each CNOT gate. Note that there are two successive H gates added in the y_1 wire between the CNOT gate controlled from x_1 and the CNOT gate controlled from x_0. The first H gate added will be used as the ending H gate in the phase kickback pattern, reversing the control position of the CNOT gate from original x_1 to y_1. The second added H gate will serve as the initial H gate in the phase kickback pattern to reverse the control position of the next CNOT gate from x_0 to y_1. The translated circuit can be found in Figure 27. It shows that x_0 and x_1 are entangled via y_1. Hence, the measured output $c_0c_1c_2$ must satisfy the property $c_0 \oplus c_1 = 0$, while c_2 is independent of c_0 and c_1.

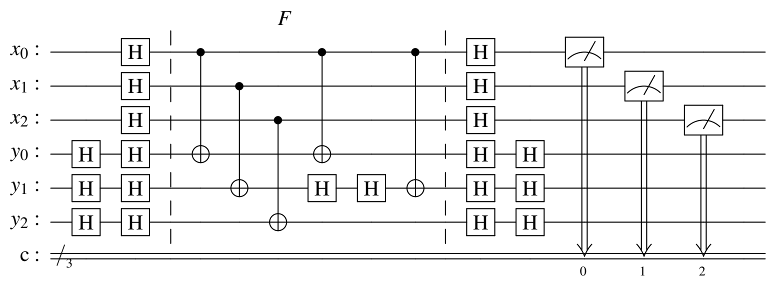


Fig. 26: Adding Successive H gates before Circuit Equivalence Translation

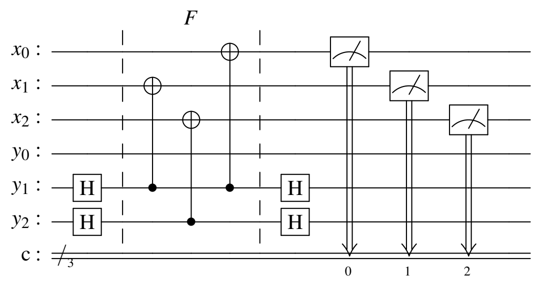


Fig. 27: Circuit Equivalence Translation via Phase Kickback in Simon's Algorithm Circuit

Note that we also need to add additional H gates in the x_k wire if there are 3 or more 1's in s. For example, if $s = 111$, two successive H gates should be inserted in the x_0 wire between the second CNOT gate and the third CNOT gate in Step 2. In general, if the secret string s has the non-zero bits indexed at locations k, l, m, $\ldots$, the measured output should satisfy the property $c_k \oplus c_l \oplus c_m \oplus \ldots = 0$. Namely, even number of these bits, c_k, c_l, c_m, $\ldots$, will have the value 1 in the measured output.

It's worth noting that the topic of constructing Simon's Algorithm oracle, while intriguing, falls beyond the scope of the textbook's contents. Its intricate details are reserved for advanced students who have already grasped the fundamental principles discussed in the labs. Whether this topic will be addressed depends on the instructor's discretion, allowing flexibility to adapt to the pace and progress of the lab.

4.7 Hands-on Lab 7: Grover's Search Algorithm

As described in the textbook, Grover's search algorithm is a quantum computing technique designed to accelerate the search for the specific item(s) in an unstructured database. In the algorithm, an initial quantum state is prepared by applying a

Hadamard transform to all possible input states. Subsequently, a specialized quantum oracle is employed to mark the target state by inverting its phase (i.e. flipping the sign of its probability amplitude). The next step involves a process of applying a diffusion operator, which amplifies the probability amplitude of the target state while reducing the amplitudes of other states. The phase inversion oracle and the amplitude amplification (also called amplitude magnification) procedure may need to be applied repeatedly approximately $\sqrt{m}$ times, where m is the number of items in the database. For example, in the case of an NP-Complete problem 3-SAT with n bits, there are $m = 2^n$ possible assignments in the database. Grover's search algorithm can reduce the run-time complexity from $O(2^n)$ to $O(\sqrt{m})$, which is equivalent to $O(2^{n/2})$. While quadratic speedup may not be as impressive as exponential speedup, it remains valuable for handling massive data sets.

The first experiment in this lab asks students to load and run a template program which finds the desired element location 11 among 2-bit strings: 00, 01, 10, and 11. In fact, this is a very simple SAT problem that determines the satisfiability of the expression $q_0 \wedge q_1$. Figure 28 shows the circuit which includes the initial Hadamard transforms, the phase inversion oracle, and the amplitude amplification. The phase inversion oracle can be realized using a Controlled-Z gate `cz(c,t)`, which flips the phase of the target qubit `t` if the control qubit `c` is in the $|1\rangle$ state. That is, the operation matrix of Controlled-Z is

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

It's symmetric and it doesn't matter which qubit is the controlled or target. As described in [15], the Controlled-Z gate is a phase-logic operation which performs phase AND (denoted as pAND). In this experiment, students are asked to use the Operator class defined in the Qiskit `quantum_info` library in their program to display the oracle operation matrix and the amplitude amplification matrix. They will find out that the operation matrix of amplitude amplification is

$$\begin{bmatrix} -0.5 & 0 & 0 & 0 \\ 0 & -0.5 & 0 & 0 \\ 0 & 0 & -0.5 & 0 \\ 0 & 0 & 0 & -0.5 \end{bmatrix}$$

and check whether it is the same as in the textbook. Students need to look at the output state vector (i.e. probability amplitude) to find the location with the highest probability.

Note that neither the textbook nor the reference book presents how to construct the amplitude amplification circuit. The geometric visualization of Grover's algorithm, as shown in Figure 29, can assist students in understanding how the algorithm works, particularly the construction of amplitude amplification. Below is a brief description. Remember that the system is initialized to the uniform superposition over all states

$$|s\rangle = H^{\otimes n}|0^n\rangle = \frac{1}{\sqrt{m}} \sum_{x=0}^{N-1} |x\rangle, \text{ where } m = 2^n$$

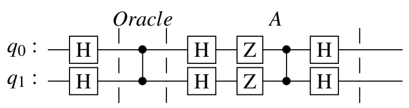


Fig. 28: Grover's Algorithm with the Boolean expression $q_0 \wedge q_1$

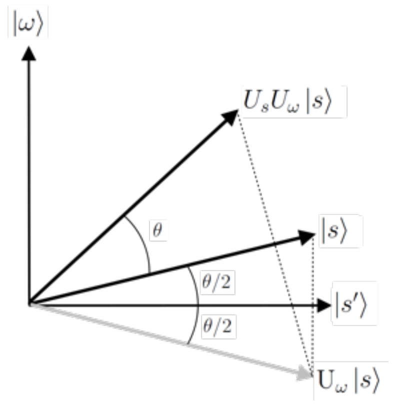


Fig. 29: Geometric interpretation of the first iteration of Grover's algorithm (Courtesy of [6])

prior to entering the phase inversion oracle. The operator of the oracle U_ω, which flips the sign of the probability amplitude associated with the target location $|\omega\rangle$, will transform the state to $U_\omega\,|s\rangle$ before amplitude magnification. The amplitude amplification operator, $U_s = 2\,|s\rangle\langle s| - I$, reflects the state about $|s\rangle$ from $U_\omega\,|s\rangle$ to $U_s U_\omega\,|s\rangle$, which is closer to the target state $|\omega\rangle$. The trick is how to solve the reflection $2\,|s\rangle\langle s|\, - I$. Because $|s\rangle = H^{\otimes n}\,|0^n\rangle$, we can firstly apply the Hadamard gates to transform $|s\rangle$ to $|0^n\rangle$, i.e.,

$$H^{\otimes n}\,|s\rangle = H^{\otimes n} H^{\otimes n}\,|0^n\rangle = |0^n\rangle.$$

Now we can do a reflection about the zero state via $2\,|0^n\rangle\langle 0^n|\, - I$, which can be implemented by using the Z gates and the Controlled-Z gate. Finally, we apply the Hadamard gates to transform it back. In summary, the amplitude amplification operation is represented as

$$U_s = 2\,|s\rangle\langle s| - I = H^{\otimes n}\,(2\,|0^n\rangle\langle 0^n|\, - I)\,H^{\otimes n},$$

which explains the amplitude amplification circuit shown in Figure 28. For advanced students, the instructor may guide them to consult the Qiskit textbook or Nielson's and Chung's book [17] for more detailed explanations.

In Experiment 2, students will work on another example searching for the location indexed at 01. Once more, this can be seen as a SAT problem that determines the satisfiability of the expression $\neg q_0 \wedge q_1$. Students can modify the previous template program by adding two X-gates, one is before the Controlled-Z and the other is after the Controlled-Z, on the qubit q_0 in the oracle. The X-gate before the Controlled-Z

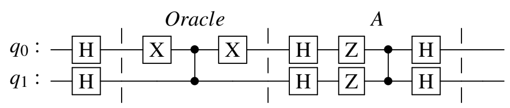


Fig. 30: Grover's Search Algorithm with the Boolean expression $\neg q_0 \wedge q_1$

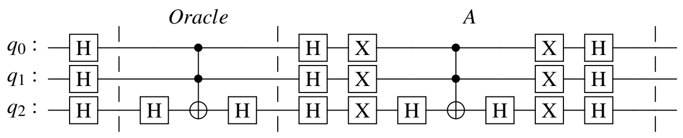


Fig. 31: Grover's Algorithm with the Boolean expression $q_0 \wedge q_1 \wedge q_2$

performs the negation of q_0, while the second X-gate uncomputes it back. Theoretically, this pair of X-gates, one swapping columns and the other swapping rows in the Controlled-Z operation matrix, moves the value -1 in the matrix to the desired element location.

In the next experiment, students will load and run a template program that includes a 3-bit oracle with the desired element location at 111. Because Qiskit does not support the ccz gate, the template program uses a ccx gate instead, but it requires placing a pair of H-gates before and after the X gate in the ccx operation. It's important to note that operation of HXH is equivalent to a Z gate. The program outputs two distinct probability amplitudes. The higher one is located at 111, while the lower one is situated at other locations. Students are asked to compare these two probability amplitudes with the two probability amplitudes mentioned in the textbook. As mentioned earlier, in order to achieve a higher probability of obtaining the correct answer, it may be necessary to apply the oracle and amplitude amplification repeatedly. To repeat the process, we do not recommend that students manually compose the oracle and amplitude amplification circuits again or use a loop construct in Python. This is because it may result in the number of gate operations growing exponentially to $O(2^{n/2})$. Therefore, students are instructed to adapt the `repeat()` method in Qiskit, which generates some kind of branch instruction like in [10] to jump back to the oracle for repetition.

In this experiment, it is also interesting for students to observe that if the oracle and the amplitude amplification tasks are performed three times, the probability amplitude of the desired location will decrease. In other words, it becomes overcooked.

We plan to add another experiment with a slightly more complicated Boolean expression, e.g. $(\neg q_0 \vee q_1) \wedge q_2$, in the future. As shown in Figure 32, after negating q_0, we use the implementation in Figure 8 to perform the OR operation, which places the result of the clause $(\neg q_0 \wedge q_1)$ into the ancilla qubit. Next, we use the Controlled-Z gate to phase AND the ancilla qubit with q_2. Finally, we have to perform uncomputation to clean up temporary effects on the ancilla bit so that it can be re-used. This experiment justifies why students need to learn how to use quantum gates to emulate

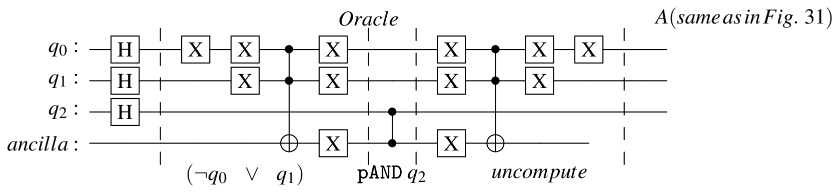


Fig. 32: Grover's Algorithm with the Boolean expression $(\neg q_0 \lor q_1) \land q_2$

classical AND and OR gates in Lab 3. An even more complicated example, which contains several clauses in a Boolean formula, can be found in [15].

5 Survey Results

We conducted an anonymous survey at the end of each lab excluding Lab 1, which focused on a review of Python rather than quantum computing. Each student received a survey form in which they were asked, for each experiment, to express agreement with the statement 'The experiment is helpful for understanding the subject'. For the Likert-like scale responses, the highest "score" was associated with the Strongly Agree response (5), and the lowest "score" was associated with the Strongly Disagree response(1). This allows participants to indicate their level of agreement. Survey responses from all experiments in each lab were combined.

Figure 33 presents the aggregated survey results via the Likert-Scale chart for each lab given to about 33 students. The responses which agree with the statement are shown to the right of the zero line, whereas those which disagree are shown to the left of the zero line. The responses which neither agree nor disagree are split down the middle. Lab 2 and Lab 3 received the lowest rating. We believe this is because students are not yet familiar with the quantum circuits. In general, the majority of responses, classified as either Strongly Agree or Agree, expressed agreement with the survey question. This indicated our hands-on labs were well-perceived by our students.

6 Conclusion and Future Work

This paper has endeavored to share our valuable teaching experiences in the realm of quantum computing. A collection of engaging laboratory experiments has been developed to serve as supplementary resources for the textbook. Many of the experiments are closely linked with the content of the textbook. These experiments capture students' interest while also offering a concrete platform for validating their comprehension through the comparison of experimental outcomes with textbook examples. The feedback collected from the students has, overall, been very positive. The students indicated that these practical labs help them not only improve their hands-on

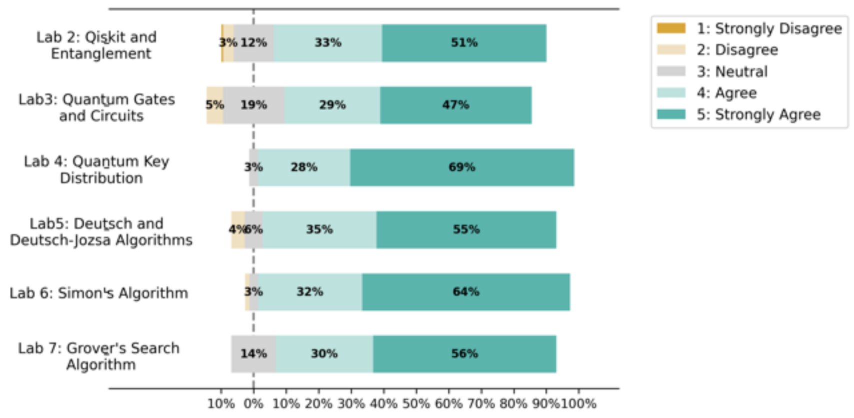


Fig. 33: Survey Results

skills in quantum programming, but also understand the rationale behind them. We believe that this report is valuable for utilization as an instructor's manual for Quantum programming education.

In the future, we plan to incorporate additional quantum programming experiments, such as the bridge gate [13], satisfiability problem with a slightly more complex Boolean formula, Shor's algorithm [26], and others, into the hands-on labs. We are also currently working on a sequel to our quantum computing class, expanding the curriculum to encompass areas such as quantum communications, quantum system software like transpilers, and advanced quantum algorithms.

Acknowledgment

Our work was supported in part by Dr. Galetto's NSF grant DMS-2200844, Dr. López's NSF grants DMS-2201094 and DMS-2401558, and the Cleveland Innovation District grant funded by JobsOhio, a private non-profit corporation.

References

1. C. Bernhardt. *Quantum Computing for Everyone*. The MIT Press, Cambridge, Massachusetts, USA, 2020.
2. F. Cardetti, N. Khamsemanan, and M. C. Orgnero. Insights Regarding the Usefulness of Partial Notes in Mathematics Courses. *Journal of the Scholarship of Teaching and Learning*, 10(1):80–92, Feb. 2012.
3. G. Carrascal, A. del Barrio, and G. Botella. First experiences of teaching quantum computing. *The Journal of Supercomputing*, Vol. 77:2770–2799, 2021.
4. Cleveland Clinic. Quantum Computing. `https://my.clevelandclinic.org/research/computational-life-sciences/discovery-accelerator/quantum-computing` (last accessed 2023-06-10).

5. E.F. Combarro, S. Vallecorsa, L. J. Rodríguez-Muñiz, A. Aguilar-González, J. Ranilla, and A. Di Meglio. A report on teaching a series of online lectures on quantum computing from CERN. *The Journal of Supercomputing*, Vol. 77:14405–14435, 2021.
6. Danski14. Own work, CC BY-SA 3.0. `https://commons.wikimedia.org/w/index.php?curid=18415805` (last accessed 2023-08-10).
7. D. Deutsch. Quantum theory, the Church-Turing Principle and the universal quantum computer. In *Proceedings of the Royal Society of London, Series A*, 1985.
8. D. Deutsch and R. Jozsa. Rapid solutions of problems by quantum computation. In *Proceedings of the Royal Society of London, Series A*, 1992.
9. A. Ernst. An overview of Quantum Comp. Frameworks. `https://www.ginkgo-analytics.com/an-overview-of-quantum-computing-frameworks/` (last accessed 2023-07-21).
10. X. Fu, L. Riesebos, M. A. Rol, Jeroen van Straten, J. van Someren, N. Khammassi, I. Ashraf, R. F. L. Vermeulen, V. Newsum, K. K. L. Loh, J. C. de Sterke, W. J. Vlothuizen, R. N. Schouten, C. G. Almudever, L. DiCarlo, and K. Bertels. eQASM: An Executable Quantum Instruction Set Architecture. In *IEEE Int'l Symposium on High Performance Computer Architectur (HPCA)*, pages 224–237, 2019.
11. L. K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the 28th Annual ACM Symposium on the Theory of Computing*, 1996.
12. IBM. Quantum System One. `https://www.ibm.com/quantum/systems` (last accessed 2023-04-10).
13. T. Itoko, R. Raymond, T. Imamichi, and A. Matsuo. Optimization of Quantum Circuit Mapping using Gate Transformation and Commutation. *Integration, the VLSI journal*, Vol. 70:43–50, 2020.
14. J. Sang and C. Yu. Hands-on Quantum Programming Labs for EECS Students. `https://arxiv.org/pdf/2308.14002.pdf` (last accessed 2023-09-29).
15. E. Johnston, N. Harrigan, and M. Gimeno-Segovia. *Programming Quantum Computers*. O'Reilly Media, Inc., Sebastopol, CA, USA, 2019.
16. J. Kiper. CSE 470N Course Syllabus. Miami University, Spring 2022.
17. M. Nielson and I. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, Cambridge, England, 2010.
18. A. Matsuo. Grover's algorithm examples: Finding solutions to 3-SAT problems. `https://github.com/Qiskit/qiskit-tutorials/blob/master/tutorials/algorithms/07_grover_examples.ipynb` (last accessed 2022-12-22).
19. M. Mykhailova and K. M. Svore. Teaching Quantum Computing through a Practical Software-driven Approach: Experience Report. In *SIGCSE '20: The 51st ACM Technical Symposium on Computer Science Education*, 2020.
20. P. Nation, H. Paik, A. Cross, and Zaira Nazario. The IBM Quantum heavy hex lattice. `https://research.ibm.com/blog/heavy-hex-lattice` (last accessed 2022-12-10).
21. Qiskit. Qiskit Textbook. `https://qiskit.org/learn` (last accessed 2023-06-10).
22. Qiskit. Quantum Computing Labs. `https://qiskit.org/learn/course/quantum-computing-labs` (last accessed 2023-06-10).
23. IBM Quantum. Develop quantum experiments in IBM quantum lab. `https://quantum-computing.ibm.com/` (last accessed 2023-06-10).
24. R. Raymond. The Simon Algorithm. `https://notebook.community/antoniomezzacapo/qiskit-tutorial/community/algorithms/simon_algorithm` (last accessed 2022-07-22).
25. Ö. Salehi, Z. Seskir, and İ. Tepe. A Computer Science-Oriented Approach to Introduce Quantum Computing to a New Audience. *IEEE Transactions on Education*, Vol. 65:1–8, 2022.
26. P.W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science, IEEE Computer Society*, 1994.
27. D.R. Simon. On the Power of Quantum Computation. *SIAM Journal on Computing*, Vol. 26, 1997.
28. The Jupyter Notebook. User Documentation. `https://jupyter-notebook.readthedocs.io/en/stable/notebook.html` (last accessed 2023-06-10).
29. Cleveland State University. News & Announcements: CSU will undertake joint interdisciplinary research and education with the Cleveland Clinic. `https://www.csuohio.edu/news/ibm-quantum-system-one-debuts-clinic-joint-research-horizon` (last accessed 2023-04-10).
30. N. S. Yanofsky and M. A. Mannucci. *Quantum Computing for Computer Scientists*. Cambridge University Press, Cambridge, England, 2008.
31. N. Yu, R. Duan, and M. Ying. Five two-qubit gates are necessary for implementing the Toffoli gate. *Physical Review A,*, Vol. 88, 2013.

32. A. Zulehner, A. Paler, and R. Wille. An Efficient Methodology for Mapping Quantum Circuits to the IBM QX Architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 1226 – 1236, 2019.