Quantum Computing and Quantum Algorithms

Daniel Serban

A Senior Thesis submitted in partial fulfillment
of the requirements for graduation
in the Honors Program
Liberty University
Spring 2020

Acceptance of Senior Honors Thesis

This Senior Honors Thesis is accepted in partial
fulfillment of the requirements for graduation from the
Honors Program of Liberty University.

____
Melesa Poole, Ph.D.
Thesis Chair

____
Scott Long, Ph.D.
Committee Member

____
David Schweitzer, Ph.D.
Assistant Honors Director

____
Date

Abstract

The field of quantum computing and quantum algorithms is studied from the ground up. Qubits and their quantum-mechanical properties are discussed, followed by how they are transformed by quantum gates. From there, quantum algorithms are explored as well as the use of high-level quantum programming languages to implement them. One quantum algorithm is selected to be implemented in the Qiskit quantum programming language. The validity and success of the resulting computation is proven with matrix multiplication of the qubits and quantum gates involved.

Quantum Computing and Quantum Algorithms

Introduction

The field of quantum computing and quantum algorithms will be explored from the ground up. The discussion of quantum computing will start with the most basic elements and concepts and work its way up from there, increasing in complexity. In this discussion, the quantum computing qubit will be introduced and contrasted from the classical bit, both in representation, usage, and properties. From there, quantum gates will be explained, showing some of the different kinds of gates that there are to work with and how they operate on qubits. After this, the discussion will move to stringing different quantum gates together to operate on the qubits in series and produce more meaningful results. This also opens the door to looking at quantum algorithms and analyzing a few of the most prominent and groundbreaking ones such as Shor's algorithm, Grover's algorithm, and the Deutsch-Jozsa algorithm (Rhami, Shamanta, & Tasnim, 2012). There will also be a discussion of the kinds of classical computing problems and algorithmic complexity that can be solved more efficiently when leveraging the type of computation and data representation available in quantum computers. Furthermore, current limitations on quantum computers due to outside interference or computational noise will be discussed, as well as methods used to try and get around this issue. One of the final topics to be discussed will be modern high-level quantum programming languages such as PyQuil, Q#, and Qiskit. A quantum algorithm will be selected and implemented with the Qiskit quantum programming language. The success and accuracy of the resulting computation will be proven by stepping through the algorithm with matrix multiplication.

Quantum Data Representation

Classical Bits

As is widely known in the field of computer science, the classical bit can hold one of two states: either a 1 or a 0. This is the basis and foundation of all of modern computing, which can involve billions of bits and bytes and billions of computations every second on only a single machine such as a desktop computer. The hardware or physical implementation and storage of such a value may differ from device to device; however, the meanings and interpretations stay the same: 1 and 0, true and false, or on and off.

Qubits

Qubits are the kind of computational bits used in the calculations of quantum computers. As opposed to the binary states of classical bits, the states represented and held by qubits are quantum mechanical and probabilistic in nature (Lahoz-Beltra, 2016). Qubits can hold a state of one, zero, or some combination of both at the same time. This characteristic of qubits necessarily makes their representation and description moderately more complicated than those of classical bits. For example, whereas a classical bit with the value of 0 can be represented with the matrix (0), a qubit with the value of 0 must be represented with the matrix $(1, 0)^T$ (Lahoz-Beltra, 2016). Due to the probabilistic nature of quantum computing, in order to accurately represent the state of a qubit, there must be probabilities indicated for both the qubit being a 0 and being a 1 when read (Lahoz-Beltra, 2016). The number in row one of the matrix for a qubit 0 represents the probability that the qubit will be 0 when read or measured, and the number in row two represents the probability that the qubit will be 1 when measured. As can be determined from the matrix for a qubit 0, there is a 100% chance that the qubit will be read as a 0 and a 0%

chance that it will be read as a 1. Likewise, (1) and $(0, 1)^T$ show the matrix representations of both a classical bit with the value of 1 and a qubit with the value of 1, respectively. The two-probability representation of qubits (for the probability of reading either a 0 or a 1 value) seems as though it is common sense or possibly even redundant.

Superposition. Obviously, if a qubit holds the value of 1, it is not going to be measured as a 0. However, the usefulness of this special representation becomes apparent when performing operations on qubits that do not give them a value of either 1 or 0, but something in between. This characteristic is both the quantum mechanical magic and power of quantum computing. When working with qubits, they can be put into a state that is known as superposition (Lahoz-Beltra, 2016). This is where the qubit has both a chance of being a 0 and a chance of being a 1 at the same time (Lahoz-Beltra, 2016). At first glance, this does not seem possible or even logical. However, at the quantum mechanical level, particles do behave in this strange yet mathematically consistent way.

Collapsing to a value. As was mentioned about qubits in superposition, they can be put into a state such that there is both a probability of being a 1 or a 0 when read. For example, a qubit can have a state that there is a 50% chance it will be a 0 and a 50% chance that it will be a 1. Another possible state is that it could be a 25% chance of a 0 and 75% chance of a 1. Whatever the superposition state may be, when the qubit finally gets read or measured, it is only ever a 1 or a 0, and no information about the preceding superposition can really be read besides that (Lahoz-Beltra, 2016; Portugal, 2018). The reason for this collapsing to one value is a quantum mechanical property that, in essence, states that whenever something in superposition is measured by a device, that state is disrupted and it collapses to either a 1 or a 0, in the case of a qubit (Botsinis, Ng, & Hanzo, 2013). If the qubit randomly collapsed to either state, this powerful aspect of quantum computing would have no real meaning and be utterly worthless. Fortunately for quantum computing and other areas of physics, the way qubits collapse to a value is very meaningful and mathematically consistent.

Consistency of quantum probabilities. The mathematical consistency of the quantum probabilities of qubits in superposition can be determined when repeatedly putting the qubits into the same states and measuring them, causing them to collapse into a 0 or 1. When done repeatedly, the probabilities involved in the superposition can be detected, as the percentage of zeros read should be consistent with the probability of the qubit being a zero and likewise for the ones. It is this consistency regarding a qubit's superposition and its corresponding collapse to a value that allows computer scientists to extrapolate a significant amount of meaning from the quantum computational data.

Before continuing, the nature of the probabilistic quantities that represent qubit states must be further explained. As was seen previously in $(0, 1)^T$, the matrix representation of a qubit with a value of 1, the state of a qubit can be represented by two quantities which describe its probability of collapsing to a zero or a one when measured (Lahoz-Beltra, 2016). A general representation of these two quantities can be signified with the variables *a* and *b*, where *a* is the probabilistic quantity for the qubit's collapse to zero and *b* is the probabilistic quantity for its collapse to one. These two quantities, *a* and *b*, do not however represent the qubit's direct probability in this regard. Instead, the probabilistic quantities, *a* and *b*, follow the mathematical equation $|a|^2 + |b|^2 = 1$ (Lahoz-Beltra, 2016). As a result, a^2, not *a*, is representative of the *actual* probability of a collapse to zero. The like goes for b^2 respective to the probability for a collapse to one. From this point forward, any use of the variables *a* or *b* within notation or equations will be only to represent the quantum probabilistic quantities for zero and one, respectively.

Complex numbers. In addition to the other mathematical properties of the probabilistic factors that represent the states of qubits, they also have an extra feature that allows even more possibilities when representing and transforming states. This extra property of these qubit quantities is that they can be complex numbers (Lahoz-Beltra, 2016). This means that the quantities represented by *a* and *b* can have both a real and imaginary component, with the imaginary component being represented by *i*.

Matrix representation. One form of qubit representation that has already been utilized previously in the discussion about qubits holding a value of 0 or 1 is matrices. Matrices can be a useful tool to indicate quantum probabilities or states since they can be used to hold two values, which are necessary for the representation of one qubit (Lahoz-Beltra, 2016). In addition, matrices can also be used to represent the state of more than one qubit at the same time, given that more than two values will be held within the matrix. Furthermore, matrices can be used to represent the changes a quantum gate will make on a qubit or on multiple qubits. Matrix multiplication can be employed, multiplying the quantum gate's matrix and the qubit's matrix together to get the resulting state of the qubit or qubits after they are transformed by the quantum gate (Lahoz-Beltra, 2016). Quantum gates perform transformations or calculations on qubits similarly to how classical gates perform transformations or calculations on classical bits. A general form of the matrix representation of a qubit is $(a, b)^T$ (Lahoz-Beltra, 2016).

Ket notation. Another way to represent the state of a qubit or system of qubits is with the use of ket notation (Botsinis et al., 2013). It is also referred to as Dirac notation (Lahoz-Beltra, 2016). The general representation of a qubit (q) using the quantities a and b is the equation $|q\rangle = a|0\rangle + b|1\rangle$ (Botsinis et al., 2013). In the equation, a ket is where there is a vertical bar and a right angle bracket surrounding a number or symbol (Botsinis et al., 2013). For example, in $|q\rangle = a|0\rangle + b|1\rangle$, the state of qubit q would read "a ket zero plus b ket one." At times when the complex states of multiple qubits need to be represented together, it can be much simpler to use ket notation as opposed to a matrix, because the size of the matrix required for accurate state representation may be quite large and difficult to work with. Many times, the much more compacted size of the same representation in ket notation tends to be the optimal choice.

Bloch sphere. Yet another way to conceptualize and depict the states of qubits is with the Bloch sphere. The Bloch sphere is a geometric way to visualize the state of a qubit (Lahoz-Beltra, 2016). The state is depicted by a vector, with its tail originating from the center of the sphere and its tip pointing out toward the surface (Botsinis et al., 2013). The positive *z*-axis of the sphere represents the qubit being in a state of 0, and the negative *z*-axis represents the qubit being in a state of 1 (Botsinis et al., 2013). Many other positions besides those two ends of the *z*-axis represent superpositions of the qubit involving real or complex quantities (Botsinis et al., 2013).

Quantum entanglement. One of the final unusual features of qubits, and what is perhaps the most perplexing of them, is quantum entanglement. Two qubits in superposition can become entangled after undergoing certain transformations in relation to each other because of specific quantum gates (Botsinis et al., 2013). What the entanglement of two qubits actually means is that, if both qubits separately have a 50% chance of being a 0 or a 1 and then they become entangled, the first qubit still has that same chance of being either 1 or 0 (Botsinis et al., 2013). However, once that first qubit is measured and the superposition collapses down to either a 1 or a 0, the other qubit that it was entangled with then has a 100% chance of being the same value as the first qubit when it gets measured (Botsinis et al., 2013). The value of one qubit becomes dependent upon and related to the value of the other, even after the collapse of the superposition. Unless there is interference from the surrounding environment, there is no chance of the two qubits not holding to this property, even over vast distances (Botsinis et al., 2013).

Difference Between Qubits and Classical Bits

The difference between qubits and classical bits is quite vast. Whether looking at their differences in physical implementation, forms of representation, or the special quantum-mechanical properties of qubits, it is apparent that the two are only loosely similar. Qubits can certainly perform the same calculations that classical bits can, but the real computing advantage arises when the special properties of qubits are fully utilized, such as the ability to take the state of a superposition of real or complex numbers or to cause quantum entanglement to occur between multiple qubits, making the outcomes of the quantum computations contain significantly more meaning than those of classical computers.

Quantum Gates and Transformations

After having finished exploring the nature of qubits, the next aspect of quantum computing that must be discussed is the quantum gate. There are many different kinds of quantum gates as well as many different physical implementations of them. However, there will be less of a focus on the physical implementation of these gates as opposed to the theoretical and mathematical aspects of them. Some quantum gates are basic and only transform one or two qubits at a time. Others are more complex and are compound gates, which are constructed from specific arrangements of several basic quantum gates. Some of the most commonly used quantum gates from both categories will be examined.

Basic Gates

Basic gates perform simple transformations on one or two qubits. These simple gates are the building blocks for more complex gates and operations, as well as quantum algorithms. One way the transformations caused by these gates on a qubit can be visualized is as rotations of the qubit's state around or across the Bloch sphere. There are some gates that rotate the qubit's state one way around the Bloch sphere, and there are some gates that rotate the state a different way. With all these transformations and rotations, the goal of all of this is to ultimately manipulate or nudge the qubit into a state such that the collapse of the superposition will yield the most meaningful results, whether the task is to search a list, factor a number, or find an optimal value.

When performing transformations or calculations on a qubit with a basic quantum gate, the resulting state of the qubit or qubits can be found through matrix multiplication (Lahoz-Beltra, 2016). Not only are qubits represented by matrices, but the effects of quantum gates can also be represented by matrices (Lahoz-Beltra, 2016). Because of this, the matrix for the quantum gate and the matrix for the qubit can be multiplied together, and the resulting matrix will be the qubit's new state (Lahoz-Beltra, 2016). There are a variety of quantum gates that have different effects on qubits. Among others, two especially important and well-known basic quantum gates are the Hadamard gate and the *CNOT* gate, which are employed in the creation of an entangled state between two qubits.

Hadamard gate. The Hadamard gate "is a generalization of the discrete Fourier transform" (Lahoz-Beltra, 2016, p. 6). It can take a qubit and put it in superposition if it was previously at a 1 or 0, and it can also take a qubit that is in superposition and bring it back down to the state of being either a 1 or a 0 (Lahoz-Beltra, 2016). The Hadamard gate is represented in Equation 1 (Lahoz-Beltra, 2016).

$$H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{1}$$

***CNOT* gate.** The *CNOT*, or controlled *NOT*, gate takes two qubits. One of the qubits will be the control bit, and if it is a 1, the value of the other qubit will be flipped (Smith, Curtis, & Zeng, n.d.). However, if the control bit is not quite 1 or 0, but it's somewhere in between, that is where the calculations get a bit more complicated and is also why these calculations are best represented with matrices and matrix multiplication. The *CNOT* gate is a 4 × 4 matrix since it has to multiply into a qubit matrix that has to account for the states of two qubits at the same time, which will cause it to be a 4 × 1 matrix instead of a 2 × 1 matrix (Botsinis et al., 2013). The *CNOT* gate has the matrix representation that is depicted in Equation 2 (Botsinis et al., 2013).

$$CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{2}$$

Pauli *X* gate. The Pauli *X* gate is also known as the *NOT* gate, in that it changes a qubit in state 1 to 0 and 0 to 1 (Lahoz-Beltra, 2016). The Pauli *X* gate has the matrix representation that is depicted in Equation 3 (Lahoz-Beltra, 2016).

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{3}$$

Pauli *Y* gate. The Pauli *Y* gate rotates the qubit such that ket 0 becomes *i* times ket 1 and ket 1 becomes -*i* times ket 0 (Lahoz-Beltra, 2016). The Pauli *Y* gate has the matrix representation that is depicted in Equation 4 (Lahoz-Beltra, 2016).

$$Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \tag{4}$$

Pauli *Z* gate. The Pauli *Z* gate rotates the qubit such that ket 0 remains ket 0 and ket 1 becomes ket -1 (Lahoz-Beltra, 2016). The Pauli *Z* gate has the matrix representation that is depicted in Equation 5 (Lahoz-Beltra, 2016).

$$Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{5}$$

Compound Gates

Compound gates are quantum gates that are comprised of potentially dozens of basic gates in series. A relatively common compound quantum gate is the Toffoli gate. The Toffoli gate, for example, is comprised of about 15 basic gates (IBM Research and the IBM QX team, 2017).

Toffoli gate. The Toffoli gate is also known as the *CCNOT* gate (Lahoz-Beltra, 2016). This compound gate is made up of many basic gates in series. The Toffoli gate works on three qubits at the same time and performs a double controlled *NOT* (Lahoz-Beltra, 2016). To put it simply, if the first two qubits are in state 1, the third qubit gets inverted (Lahoz-Beltra, 2016). The Toffoli gate is the quantum equivalent of the classical *AND* gate (IBM Research and the IBM QX team, 2017). An added benefit of the Toffoli gate is that it is reversible, so no information about how the result was achieved is lost because of the gate (IBM Research and the IBM QX team, 2017). Because the Toffoli gate works on three qubits at the same time, its matrix representation is significantly larger than the previous gates' matrices. The Toffoli gate has the matrix representation that is depicted in Equation 6 (Lahoz-Beltra, 2016). The Toffoli gate is represented by an 8 × 8 matrix because it will be multiplied with an 8 × 1 matrix representing the probabilities of each of the eight possible resulting binary strings of the three qubits. This 8 × 1 matrix that represents each of the eight possible resulting binary strings of the three qubits can be found by taking the tensor product of each qubit's respective 2 × 1 matrix representation (Portugal, 2018).

$$TOF = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \qquad (6)$$

Quantum Algorithms

When working with quantum gates, the “magic” (the useful quantum physics) only really happens when stringing several of them together in special combinations to solve problems. This is how quantum algorithms are created.

There are quite a few important and groundbreaking quantum algorithms that have been created. The key to understanding these algorithms is to find the special techniques the creators used to take their qubits from a significant number of varying states to collapsing those states down to a single answer that they were looking for, with all of the other incorrect values interfering with each other and cancelling out (Rahmi et al., 2012). When done correctly, the creation of a quantum algorithm can give the user a significant advantage when it comes to solving a certain problem over another person trying to solve that same problem using a classical computer. The main drawback of these quantum algorithms is that much of their practicality or usefulness is still theoretical. This is because quantum computers today are not yet sophisticated enough to run quantum computations successfully with a large number of qubits.

Limitations of Physical Quantum Hardware

Even though current quantum computers suffer from a major source of error in computations, as will be discussed, there is also a system in place to account for this problem with the goal of fixing it. However, this solution is not perfect and has its drawbacks. Perhaps someday, quantum technology will advance to the point where these errors are no longer an issue.

Quantum noise. Quantum noise, or decoherence, during computations is the major source of error for quantum computers (Cross, Smith, & Smolin, 2015). The slightest disturbance to the qubits during the calculation could alter the state of any qubit and corrupt the whole calculation. Furthermore, it is difficult to keep qubits isolated from outside interference while also controlling them to perform quantum calculations (Preskill, 2018). In addition, every extra gate or qubit added to the quantum circuits for computations causes the probabilities for noise-related error to increase significantly (Preskill, 2018). As a result, it is very difficult to scale up quantum computers to the size and number of qubits necessary to perform meaningful and practical computations other than a small-scale proof of concept (Preskill, 2018). There are measures that have been taken to try to fix this problem, however, they are far from ideal. Until the quantum technology improves significantly in its ability to shield the qubits from noise-related error in the computations, there is a hard limit on the total number of qubits that can be effectively used in quantum computations.

Error correction. One significant measure that has been taken to counteract noise-related error in quantum computations is exploration and research into a form of error correction. As nice as this feature sounds for quantum computers, it is far from ideal. The reason is that implementing effective error correction in a quantum computer would come with a significant increase in the number of qubits and quantum gates needed for a circuit, in which the gates implementing the quantum algorithm reside (Preskill, 2018). Furthermore, these extra qubits and gates would be in the form of extra overhead for the computation, as they would be redundant in terms of the gates and qubits needed solely to perform the quantum computation in a perfect system (Preskill, 2018).

Computational Complexity

When running algorithms and comparing problems that algorithms solve, or at least attempt to solve, there are several nested classes of problems that exist (Aaronson, 2008). The first class is P, which stands for polynomial time (Aaronson, 2008). That means it takes polynomial time for a computer to solve that problem; they are generally easy for computers to solve (Aaronson, 2008). The class surrounding P is NP, which stands for nondeterministic polynomial time (Aaronson, 2008). With problems classified as NP, as size *n* increases, the number of computations may increase exponentially (Aaronson, 2008). Also within NP, but separate from P is NP-Complete (Aaronson, 2008). NP-Complete problems are the most difficult of the NPs to solve (Aaronson, 2008). There is no algorithm that exists that can solve an NP-Complete problem effectively (Aaronson, 2008). When it comes to quantum computers however, they are in a class of their own. Quantum computers are in the class BQP, which stands for bounded-error quantum polynomial time (Aaronson, 2008). This class of problems is in P, some of NP, and even outside of NP (Aaronson, 2008). However, BQP does not intersect with the NP-Complete problems (Aaronson, 2008).

Entanglement of Two Qubits

One useful algorithm that gets utilized within other larger algorithms is the entangling of two qubits. This algorithm will end with the final state of one qubit being entirely dependent on the collapse of the superposition of the other qubit (Botsinis et al., 2013). The binary string of the two qubits at the end of this algorithm will either be 00 or 11 depending on how they collapse (Botsinis et al., 2013).

The algorithm starts with both qubits in state 0 (Botsinis et al., 2013). Then, a Hadamard gate is applied to the first qubit, which will put it into superposition (Botsinis et al., 2013). After that, a *CNOT* gate will be applied to both qubits, with the first qubit acting as the control bit (Botsinis et al., 2013). If this algorithm is being calculated by hand with matrices, the tensor product of qubit one and qubit two will have to be taken, which will individually multiply each qubit matrix element by each of the opposing matrix's elements. This will form a 4 × 1 matrix representing the state of both qubits (i.e. the probability that they will be 00, 01, 10, or 11) so that it can be multiplied by the 4 × 4 *CNOT* matrix (Portugal, 2018). The resulting 4 × 1 matrix after applying the *CNOT* should indicate that there is now, due to entanglement from choosing the qubit in superposition to be the control bit, a 50% chance of the bit string being 00 and a 50% chance of the bit string being 11, with no other alternatives (Botsinis et al., 2013).

Prominent Quantum Algorithms

Shor's algorithm. Shor's algorithm was created by Peter Shor in 1994 (as cited in Rahmi et al., 2012). The main activity of Shor's algorithm is to find the factors of composite numbers in a very small number of steps (Rahmi et al., 2012). Interestingly enough, many of the steps in Shor's algorithm are actually performed on a classical computer (Rahmi et al., 2012). There is only a certain part in which the special quantum-mechanical qualities of qubits are needed to cut down on the necessary number of computations (Rahmi et al., 2012). The part of Shor's algorithm that actually uses quantum computing is where the quantum Fourier transform is used to find the period of a^x mod n (Rahmi et al., 2012). Shor's algorithm is probabilistic in that it needs to be run multiple times for guaranteed accuracy (Bruß & Leuchs, 2016). Shor's algorithm is also pretty efficient in that it runs in log *n* time (Bruß & Leuchs, 2016).

Shor's algorithm is very good at quickly finding the two prime factors of very large composite numbers used in RSA encryption (Smolin, Smith, & Vargo, 2013). When the hardware of quantum computers finally becomes sophisticated enough that it can work with integers as large as those that are used in RSA encryption, Shor's algorithm will essentially degrade any notion of privacy that users can have when sending data across the internet using this current system of encryption.

Grover's algorithm. Grover's algorithm was created by Lov Grover in 1996 (as cited in Rahmi et al., 2012). Grover's algorithm is used to search an unordered list in square root of *n* time (Montanaro, 2015). The algorithm must be run over several iterations because it is probabilistic in nature (Rahmi et al., 2012). As a result, Grover's algorithm only gives the correct answer most of the time, not all of the time (Rahmi et al., 2012). Grover's algorithm potentially offers a quadratic speedup for any algorithm it is used in that requires searching (Bruß & Leuchs, 2016).

Deutch-Jozsa algorithm. The Deutch-Jozsa algorithm was created by David Deutch and Richard Jozsa in 1992 (as cited in Rahmi et al., 2012). The Deutch-Jozsa algorithm was one of the first examples of an algorithm that was more efficient than a classical algorithm with similar objectives (Rahmi et al., 2012). Curiously, the algorithm was invented more as a thought experiment or a proof of concept, as the algorithm does not serve any real function or meet a need of any kind in the world of computing (Rahmi et al., 2012).

The algorithm and associated scenario goes as follows: There is a black box with an unknown function inside that takes inputs and gives outputs. The function either gives the same output every time no matter what the input is, or the function gives one of two different outputs depending upon the input (Rahmi et al., 2012). The task is to use quantum computing to test the box with qubits and determine which kind of function is in the box (Rahmi et al., 2012).

The beauty of the Deutch-Jozsa algorithm is that, while a classical computer can derive the quality of the mysterious function in the box in as little as two queries, a quantum computer can determine it in only one query (Rahmi et al., 2012).

Other Potential Uses

One possible use of quantum computing is matrix inversion. There is the potential that quantum computing could provide an exponential speedup when it comes to matrix inversion of large matrices (Preskill, 2018). There is another possible benefit of quantum computing in the area of deep learning. Quantum computers could potentially be very successful when it comes to deep learning involving data sets that are quantum mechanical themselves (Preskill, 2018). One other potential area where quantum computers could be useful is quantum annealing (Preskill, 2018). Quantum annealers can be used to solve optimization problems; however, they may not outperform classical computers at these problems unless the quantum technology advances and becomes significantly more resilient to outside error caused by quantum noise (Preskill, 2018).

High-Level Quantum Programming

As within classical programming, there is the trend of computer scientists to develop more and more programming languages that build off of each other and also increase the level of abstraction between the code and the metal of the machine. Furthermore, when a programming language is high-level as opposed to low-level, it is often more readable and more user-friendly toward the programmer. However, as the level of abstraction and readability increases, the degree of efficiency of the execution of the underlying code may consequently decrease, since the programmer doesn’t have direct access to the lower layers of the code. These same concepts seem to generally be true within the world of quantum programming.

In quantum programming, there are some languages that are quite cryptic and seem to be just a step above assembly language. However, these languages, such as Quil (quantum instruction language), have many features that can be taken advantage of, such as a large variety of gates to choose from in quantum circuit construction or the ability to define either new gates or existing gates that are not already implemented (Smith et al., n.d.). Other quantum programming languages, such as ones that are C-based or Python-based, may be more readable and easy to program, but they might not offer all the gates or features a programmer is looking for.

Good Features for Quantum Programming Languages

Some suggested features for good quantum programming languages are as follows: Quantum programming languages should allow for descriptions of algorithms both as a series of quantum gates as well as through mathematical formulas (Valiron, Ross, Selinger, Alexander, & Smith, 2015). Allocation and measurement should be allowed for registers (Valiron et al., 2015). The language should be able to estimate the required resources for code, such as the number of qubits (Valiron et al., 2015). Finally, the language should put in place some allotment of resources for use regarding errors and quantum error correction functionality (Valiron et al., 2015).

Prevalent Languages

Qiskit language. Qiskit (quantum information software kit) can be Python-based, JavaScript-based, or Swift-based (LaRose, 2019). It is managed by IBM (LaRose, 2019). Qiskit offers users support for a nice variety of quantum algorithms to use in programs (LaRose, 2019). Quantum computer simulators are available through the language for users to run locally on their computer (LaRose, 2019).

PyQuil language. PyQuil is a Python-based higher-level language for the low-level Quil (LaRose, 2019). It is managed by Rigetti (LaRose, 2019). PyQuil seems to support a wide variety of quantum algorithms for programmers to use (LaRose, 2019). To run code on a quantum computer simulator, it requires an API key and a remote connection to Rigetti's hosted simulators (LaRose, 2019).

Q# language. Q# is a standalone programming language that looks a lot like C# (LaRose, 2019). Q# is managed by Microsoft (LaRose, 2019). It appears that Q# has support for several quantum algorithms that both Qiskit and PyQuil do not have support for (LaRose, 2019). The Q# language also offers support for a local quantum computer simulator on which to run code, as opposed to requiring users to connect to a remote simulator (LaRose, 2019).

Quantum Algorithm High-Level Implementation

The quantum algorithm to entangle two qubits will now be implemented in a high-level quantum programming language. The high-level language to be used is the Python-based language Qiskit. The Qiskit code to implement this algorithm will be written and run. A diagram of the quantum circuit created in Qiskit will be displayed as well as the results of the execution on a simulator of a quantum computer. In order to make sure the results are accurate, the steps of the matrix multiplication to arrive at the final answer or results will be shown as well.

Figure 1 shows the sequence of matrix multiplications required to calculate the effects of the quantum entanglement algorithm. Figure 2 shows the Qiskit code implementation of that same algorithm/quantum circuit.

$$\begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix} \times \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix}$$

Step 1: Applying Hadamard gate to first qubit (state 0) to gain superposition.

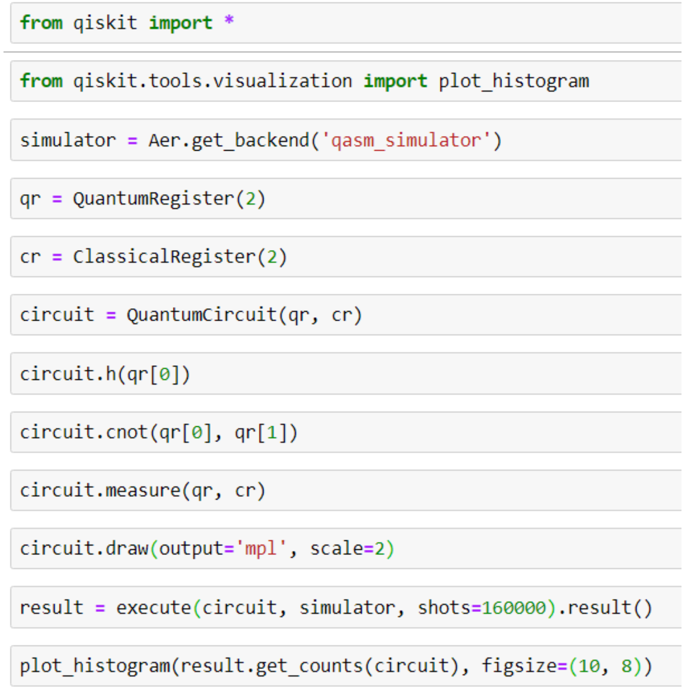
$$\begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ \frac{1}{\sqrt{2}} \\ 0 \end{pmatrix}$$

Step 2: Taking tensor product of qubit one and qubit two for combined states.

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \times \begin{pmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ \frac{1}{\sqrt{2}} \\ 0 \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ 0 \\ \frac{1}{\sqrt{2}} \end{pmatrix}$$

Step 3: Applying *CNOT* gate to qubits one and two, getting final state with a 50% chance of the bit string being 00 and a 50% chance of it being 11.

Figure 1. Matrix multiplication proof of quantum entanglement algorithm results.



```
from qiskit import *
from qiskit.tools.visualization import plot_histogram
simulator = Aer.get_backend('qasm_simulator')
qr = QuantumRegister(2)
cr = ClassicalRegister(2)
circuit = QuantumCircuit(qr, cr)
circuit.h(qr[0])
circuit.cnot(qr[0], qr[1])
circuit.measure(qr, cr)
circuit.draw(output='mpl', scale=2)
result = execute(circuit, simulator, shots=160000).result()
plot_histogram(result.get_counts(circuit), figsize=(10, 8))
```

Figure 2. Qiskit code for quantum entanglement algorithm. Two quantum registers, or qubits, are created to work with in the algorithm, and two classical registers are created to hold the collapsed values of the qubits when completed. A quantum circuit is created with which to hold the quantum gates and manipulate the qubits and classical bits. Next, a Hadamard gate is added to the circuit with `h()` and it is applied to quantum register 0 (the first qubit). After that, a *CNOT* gate is added to the circuit with `cnot()` and it is applied to quantum registers 0 and 1 (the first and second qubit). Finally, a measure gate is added to the circuit that will measure the values of the two qubits, causing their superpositions to collapse, and the collapsed values from the qubits will be stored in their respective classical registers.

`Circuit.draw()` was used to create the circuit diagram in Figure 3. The quantum circuit was executed on a simulator 160,000 times and the outcomes of the probabilities are tallied in Figure

4, showing that approximately 50% of the time, the qubits collapsed to 00, and 50% of the time, they collapsed to 11.

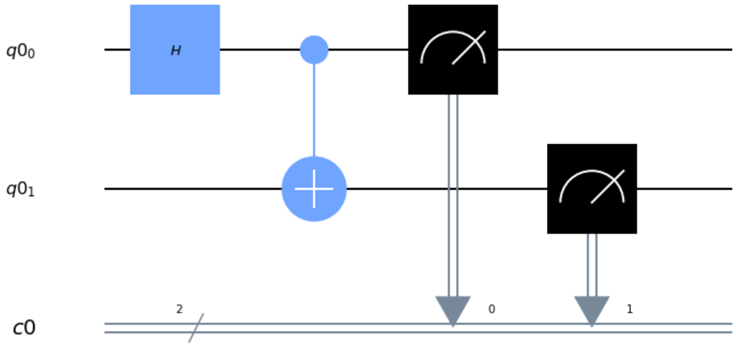


Figure 3. Diagram of quantum entanglement circuit.

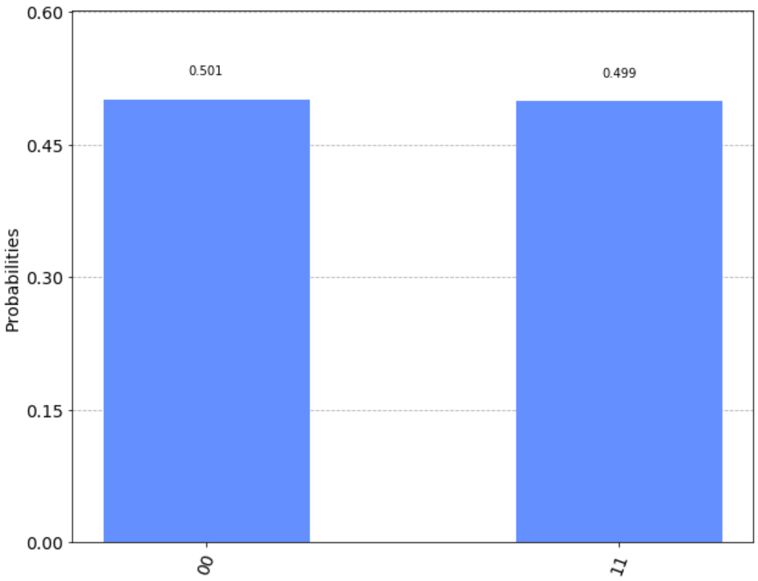


Figure 4. Results from repeated executions of quantum entanglement circuit.

Conclusion

Quantum computing and quantum algorithms have been studied from the ground up, starting with qubits, then quantum gates, quantum algorithms, and quantum programming languages. Important connections between the many parts of quantum computing have been analyzed and related to each other. The many mathematical, probabilistic, and quantum-mechanical properties of qubits have been researched and utilized. The matrix representations of qubit states and quantum gates were instrumental in proving the success and legitimacy of the computations that resulted from the use of the Qiskit quantum programming language. This subject area of quantum computing is rich in discoveries and innovation just waiting to be explored. There is no telling what the field of computer science could look like in 20 years.

References

Aaronson, S. (2008). The limits of quantum. *Scientific American, 298*(3), 62-69. doi:10.1038/scientificamerican0308-62

Berry, D. W., Childs, A. M., Cleve, R., Kothari, R., & Somma, R. D. (2015). Simulating Hamiltonian dynamics with a truncated Taylor series. *Physical Review Letters, 114*(9), 90502-90507. doi:10.1103/PhysRevLett.114.090502

Bocharov, A., & Svore, K. M. (2012). Resource-optimal single-qubit quantum circuits. *Physical Review Letters, 109*(19), 190501-190506. doi:10.1103/PhysRevLett.109.190501

Botsinis, P., Ng, S. X., & Hanzo, L. (2013). Quantum search algorithms, quantum wireless, and a low-complexity maximum likelihood iterative quantum multi-user detector design. *IEEE Access, 1*, 94-122. doi:10.1109/ACCESS.2013.2259536

Bruß, D., & Leuchs, G. (Eds.). (2016). *Quantum information: From foundations to quantum technology applications* (Vol. 1 & 2). doi:10.1002/9783527805785

Cross, A. W., Smith, G., & Smolin, J. A. (2015). Quantum learning robust against noise. *Physical Review A, 92*(1), 12327-12333. doi:10.1103/PhysRevA.92.012327

Di, Y., & Wei, H. (2013). Synthesis of multivalued quantum logic circuits by elementary gates. *Physical Review A, 87*(1), 12325-12334. doi:10.1103/PhysRevA.87.012325

Dueck, G. W., & Miller, D. M. (Eds.). (2013). *Reversible computation: 5th international conference.* doi:10.1007/978-3-642-38986-3

IBM Research and the IBM QX team. (2017). Basic circuit identities and larger circuits. Retrieved May 2, 2019, from https://quantum-computing.ibm.com/docs/guide/q-algos/basic-circuit-identities-and-larger-circuits

Lahoz-Beltra, R. (2016). Quantum genetic algorithms for computer scientists. *Computers, 5*(4), 1-31. doi:10.3390/computers5040024

LaRose, R. (2019). Overview and comparison of gate level quantum software platforms. *Quantum, 3*, 130-154. doi:10.22331/q-2019-03-25-130

Lloyd, S., Mohseni, M., & Rebentrost, P. (n.d.). Quantum algorithms for supervised and unsupervised machine learning. Retrieved May 2, 2019, from https://arxiv.org/abs/1307.0411v2

Montanaro, A. (2015). Quantum algorithms: An overview. *NPJ Quantum Information, 2*, 1-17. doi:10.1038/npjqi.2015.23

Musz, M., Kuś, M., & Życzkowski, K. (2013). Unitary quantum gates, perfect entanglers and unistochastic maps. *Physical Review A, 87*(2), 22111-22123. doi:10.1103/PhysRevA.87.022111

Odeh, A., Elleithy, K., Almasri, M., & Alajlan, A. (2013). Sorting n elements using quantum entanglement sets. *Third International Conference on Innovative Computing Technology (INTECH 2013)* (pp. 213-216). doi:10.1109/INTECH.2013.6653693

Pang, C., Zhou, R., Ding, C., & Hu, B. (2013). Quantum search algorithm for set operation. *Quantum Information Processing, 12*, 481-492. doi:10.1007/s11128-012-0385-8

Portugal, R. (2018). *Quantum walks and search algorithms* (2nd ed.). doi:10.1007/978-3-319-97813-0

Preskill, J. (2018). Quantum computing in the NISQ era and beyond. *Quantum, 2*, 79-98. doi:10.22331/q-2018-08-06-79

Rahmi, M., Shamanta, D., & Tasnim, A. (2012). Basic quantum algorithms and applications. *International Journal of Computer Applications, 56*(4), 26-31. doi:10.5120/8880-2868

Smith, R. S., Curtis, M. J., & Zeng, W. J. (n.d.). A practical quantum instruction set architecture. Retrieved May 2, 2019, from https://arxiv.org/abs/1608.03355v2

Smolin, J. A., Smith, G., & Vargo, A. (2013). Oversimplifying quantum factoring. *Nature, 499*, 163-165. doi:10.1038/nature12290

Valiron, B., Ross, N. J., Selinger, P., Alexander, D. S., & Smith, J. M. (2015). Programming the quantum future. *Communications of the ACM, 58*(8), 52-61. doi:10.1145/2699415