Experimental Demonstrations of Quantum Computing

Utilizing the IBM 5Q SDK and API

Umar Mahazudin
Ryan Shanks
Gareth Sharpe

Table of Contents

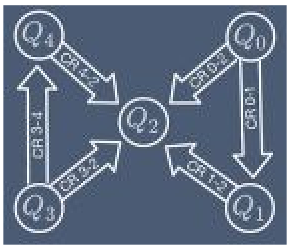
Introduction

The newly emerging paradigm shift within computing towards the vastly unexplored field of quantum computation promises a drastic transformation in the ways in which we process information. This 'quantum revolution' promises to spur the next era of innovation as quantum computers poise to tackle important problems whose complexity is exponential in nature or whose potential answers are far too numerous to be computed traditionally. This revolution, however, has significant challenges to overcome. Questions arise about the engineering feasibility of connecting thousands of qubits together, fabricating them in the first place, being able to control them, and even measuring a qubit. Even considering this were all possible, how does the average innovator or problem solver attain the means of access to a $15 million quantum computer? This paper's aim is to demonstrate that not only is this all possible, it is possible for you - the 'quantum revolution' is starting in the cloud.

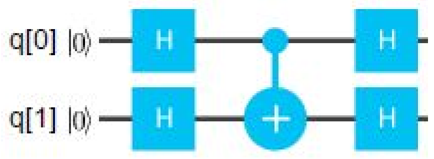
International Business Machines (IBM) is at the forefront of the emerging quantum systems. While other companies offer non-universal quasi-quantum computers, IBM has developed a solution to the availability problem of a truly universal quantum computer. Since its launch less than a year ago, about 40,000 users have run over 275,000 experiments on the IBM Quantum Experience offered free of charge to both experts and amateurs alike On the IBM Cloud. IBM has recently released a new and user-friendly application programming interface (API) which enables software developers and programmers who are short $15 million to design and develop applications using its existing five qubit IBM Cloud-based quantum computer. Not only has an API been developed, but an open-source machine code for programming the IBM quantum computer has been released to allow for more complex operations. This machine code, named OPENQASM, is in it's second stage of development and has recently been provided a thin yet functional Python wrapper to allow for operations to be integrated with other programs to further increase the available potential of the quantum computer. This paper seeks to explore the underlying functionality of the machine code behind the IBM Quantum Experience, as well as create and compute numerous quantum algorithms and test for consistency between the theoretical mathematics and the empirical results.

Challenges with the 5Q

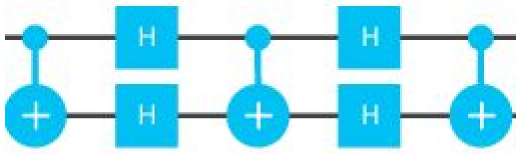
The 5Q is a small quantum computer, consisting of 5 qubits and no other accessible storage. The 5Q does not have every gate we discussed in class, for example there is no CZ gate, SWAP, or Toffoli gate. A Toffoli gate is also known as CCNOT, it flips a qubit if both of the control qubits are 1. Additionally, not all qubits are connected for CNOT, and none are connected in both directions. The 5 qubits are connected like this:



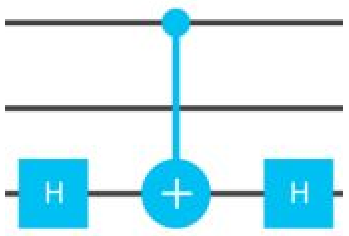
Arrows denote which way qubits can control which, so for example, q[1] can control q[2] via CNOT, and q[1] can be controlled by q[0], but q[0] cannot be directly controlled by any other qubit. Fortunately, it is possible to reverse the direction of control using Hadamard gates, so q[0] actually can be controlled by q[1] and q[2]. For example, to control q[0] with q[1], a sub circuit $H_1H_0CNOTH_1H_0$ can be built like this:



It is also not possible to control q[1] from q[3]. In order to use a qubit stored in q[3] as control for q[1], one of those would first need to be swapped into q[2]. Since there is no swap gate, we need to use the following sequence of gates to swap qubits, as described in assignment 3 (and then modified with hadamard to allow the middle CNOT to be reversed).



The 5Q also does not have a CZ gate, but one can be constructed like this, where the bottom qubit is being flipped if the top qubit is 1:



According to the wikipedia article entitled Quantum gates, any single qubit gate (which is a 2x2 unitary matrix), can be made into a controlled gate by starting with a 4x4 identity matrix and replacing the bottom - right quadrant with the matrix for the single qubit gate. We can use this to show that $HCNOTH = CZ$:

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

Note: the first and last matrices on the left hand side of the above equation represent Hadamard gates on the second qubit in a 2 qubit state. They are constructed from $I \otimes H$.

Unitary Gate Decomposition and Implementation

The IBM 5Q provides many standard gates that can be used to implement quantum circuits. These include the identity gate, the Pauli X, Y and Z gates, the Hadamard gate, the phase gate (S), the $\pi/8$ gate (T) and a controlled-not gate (CNOT). However, many complex quantum circuits require gates that are not any of these provided gates or cannot be expressed as a combination of them.

Fortunately, there is a theorem that resolves this issue. Suppose U is a unitary operation on a single qubit. Then there exist real numbers α, β, γ and δ such that $U = e^{i\alpha}R_z(\beta)R_y(\gamma)R_z(\delta)$. For a general unitary operation on a single qubit the IBM 5Q allows an arbitrary single qubit unitary gate to be implemented and used as part of a quantum circuit. It is built in as the $U3$ gate and parameterized as

$$U(\theta,\phi,\lambda) := R_z(\phi)R_y(\theta)R_z(\lambda) = \begin{pmatrix} e^{-i(\phi+\lambda)/2}\cos\frac{\theta}{2} & -e^{-i(\phi-\lambda)/2}\sin\frac{\theta}{2} \\ e^{i(\phi-\lambda)/2}\sin\frac{\theta}{2} & e^{i(\phi+\lambda)/2}\cos\frac{\theta}{2} \end{pmatrix}$$

where $R_y(\theta) = e^{-i\theta Y/2}$ and $R_z(\phi) = e^{-i\phi Z/2}$. If U can be specified with only 2 parameters then the gate can be implemented with the built in $U2$ gate and, similarly, the $U1$ can gate can be used if U can be specified with only 1 parameter.

As a corollary, there exist unitary operators A, B and C on a single qubit such that $ABC = I$ and $U = e^{i\alpha}AXBXC$ where α is some overall phase factor. By choosing $A \equiv R_z(\beta)R_y(\gamma/2)$, $B \equiv R_y(-\frac{\gamma}{2})R_z(-\frac{\delta+\beta}{2})$ and $C \equiv R_z(\frac{\delta-\beta}{2})$ and using the identities $X^2 = I$, $XYX = -Y$ and $XZX = Z$, it can be shown that $ABC = I$ and $U = e^{i\alpha}AXBXC$. This becomes important when implementing controlled unitary operations which require 2 qubits. By replacing the Pauli X gate with a CNOT gate, these set of operations implement a controlled-U operation.

For example, consider the controlled phase shift gate R_ϕ which is important in the implementation of the quantum Fourier transform. It can be shown that

$$R_\phi = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\phi} \end{pmatrix}$$

$$U = R_\phi = e^{i\frac{\phi}{2}}R_z(0)R_y(0)R_z(\phi) \Rightarrow \alpha = \frac{\phi}{2}, \beta = 0, \gamma = 0, \delta = \phi$$

Then $A = I$, $B = R_z(-\frac{\phi}{2})$, $C = R_z(\frac{\phi}{2})$. To implement the controlled phase shift gate using OPENQASM, note that $R_z(\lambda) = U(0,0,\lambda) = u1(\lambda)$. So the controlled phase shift gate $cR_\phi = cu1$ can be written as:

```
gate cu1 (lambda) a,b
{
   u(lambda/2) b;
   cx a,b;
   u(-lambda/2) b;
   cx a,b;
   u(lambda/2) a;
}
```

Grover's Search Algorithm

One of the greatest advantages to a quantum computer, and arguably one of the most desired, is a quantum computer's ability to search a database quickly and efficiently. The search algorithm developed by Lov Grover in 1996 provides a much-improved O $(\sqrt{n})$ solution to a problem which is classically an O (n) solution for an unsorted database and an O $(\log n)$ complexity for a sorted database, demonstrating a quadratic speedup. Like most other quantum algorithms, Grover's search algorithm is probabilistic solution, as will be demonstrated in the empirical testing. As this algorithm gives the correct answer with a probability less than 1, it is required that this algorithm be repeated several times before confidence can be placed in the algorithm's results. The math behind the algorithm, however, only need be demonstrated once. Other applications of the Grover Search Algorithm include detecting a prime number, estimating the mean and median of large sets of numbers, and even brute forcing passwords.

The following is a description of the steps required:

1. Initialized the procedure with uniform superposition of n qubits using H gates
2. Apply a reflection state $U = f(x)$ to identify (tag) the qubit to search for
3. Apply Grover's Search Algorithm:
    a. Apply an H gate to n qubits
    b. Apply an X gate to n qubits
    c. Apply an additional reflection state U gate corresponding to $U = 2|\Psi\rangle\langle\Psi| - 1$
        i. Note: two reflections always correspond to a rotation
    d. Apply an X gate to n qubits
    e. Apply an H gate to n qubits
4. Measure n qubits

Consider this algorithm on the smallest possible implemented circuit involving 2 qubits:

$$N = 2^n = 2^2 = 4$$

$\therefore$ There are four possible reflection states $U = f(x)$

Before any reflection states can be determined, a uniform superposition $|\Psi\rangle$ of $n = 2$ qubits.

$$H|00\rangle = \tfrac{1}{2}(|0\rangle + |1\rangle)(|0\rangle + |1\rangle) = \tfrac{1}{2}(|00\rangle + |01\rangle + |10\rangle + |11\rangle) = |\Psi\rangle$$

Using the quantum state $|\Psi\rangle$, we then apply a reflection state $U = f(x)$. Suppose we are interested in searching for the element in the 4th (last) position of our data set N. The reflection $U = f(x)$ corresponding to the fourth position can be described as:

$$U = f(x) = H_2 C_{12} H_2$$

Thus:

$$U|\Psi\rangle = H_2 C_{12} H_2 |\Psi\rangle$$

This results is:

$$H_2|\Psi\rangle = \tfrac{1}{2}\tfrac{1}{\sqrt{2}}\,(|0\rangle(|0\rangle + |1\rangle) + (|0\rangle(|0\rangle - |1\rangle) + (|1\rangle(|0\rangle + |1\rangle) + (|1\rangle(|0\rangle - |1\rangle)$$
$$= \tfrac{1}{2\sqrt{2}}(|00\rangle + |01\rangle + |00\rangle - |01\rangle + |10\rangle + |11\rangle + |10\rangle - |11\rangle) = \tfrac{1}{2\sqrt{2}}(2|00\rangle + 2|10\rangle) = \tfrac{1}{\sqrt{2}}(|00\rangle + |10\rangle)$$

$$C_{12}H_2|\Psi\rangle = \tfrac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$$

$$H_2 C_{12} H_2|\Psi\rangle = \tfrac{1}{2}(|0\rangle(|0\rangle + |1\rangle) + (|1\rangle(|0\rangle - |1\rangle) = \tfrac{1}{2}(|00\rangle + |01\rangle + |10\rangle - |11\rangle) = |\Psi'\rangle$$

As can be observed by the resulting $|\Psi'\rangle$, the state of the qubit in position four has been identified (tagged) using amplitude amplification. The result is a tag which the Grover's Search

Algorithm can identify as the desired outcome of the search. The final part of the algorithm is to apply Grover's Search. Given our 'tagged' value, we now perform a final reflection U to complete the rotation and strengthen the amplitude of our desired outcome $a = |11\rangle$ while at the same time decreasing the amplitude of all non-desired outcomes. Grover's Algorithm on $n = 2$ can be described as:

$$U = f(x) = HXH_2C_{12}H_2XH$$

Thus:

$$U|\Psi'\rangle = HXH_2C_{12}H_2XH|\Psi'\rangle$$

The full mathematical result is:

$|\Psi'\rangle = \frac{1}{2}(|00\rangle + |01\rangle + |10\rangle - |11\rangle)$

$H|\Psi'\rangle = \frac{1}{4}((|0\rangle + |1\rangle)(|0\rangle + |1\rangle) + (|0\rangle + |1\rangle)(|0\rangle - |1\rangle) + (|0\rangle - |1\rangle)(|0\rangle + |1\rangle) - (|0\rangle - |1\rangle)(|0\rangle - |1\rangle))$
$= \frac{1}{4}(|00\rangle + |01\rangle + |10\rangle + |11\rangle + |00\rangle - |01\rangle + |10\rangle - |11\rangle + |00\rangle + |01\rangle - |10\rangle - |11\rangle - |00\rangle + |01\rangle + |10\rangle - |11\rangle$
$= \frac{1}{4}(2|00\rangle + 2|01\rangle + 2|10\rangle - 2|11\rangle) = \frac{1}{2}(|00\rangle + |01\rangle + |10\rangle - |11\rangle$

$XH|\Psi'\rangle = \frac{1}{2}(|11\rangle + |10\rangle + |01\rangle - |00\rangle)$

$H_2XH|\Psi'\rangle = \frac{1}{2\sqrt{2}}(|1\rangle(|0\rangle - |1\rangle) + (|1\rangle(|0\rangle + |1\rangle) + (|0\rangle(|0\rangle - |1\rangle) - (|0\rangle(|0\rangle + |1\rangle)$
$= \frac{1}{2\sqrt{2}}(|10\rangle - |11\rangle - |10\rangle + |11\rangle + |00\rangle - |01\rangle - |00\rangle - |01\rangle) = \frac{1}{2\sqrt{2}}(2|10\rangle - 2|01\rangle) = \frac{1}{\sqrt{2}}|10\rangle - |01\rangle$

$C_{12}H_2XH|\Psi'\rangle = \frac{1}{\sqrt{2}}|11\rangle - |01\rangle$

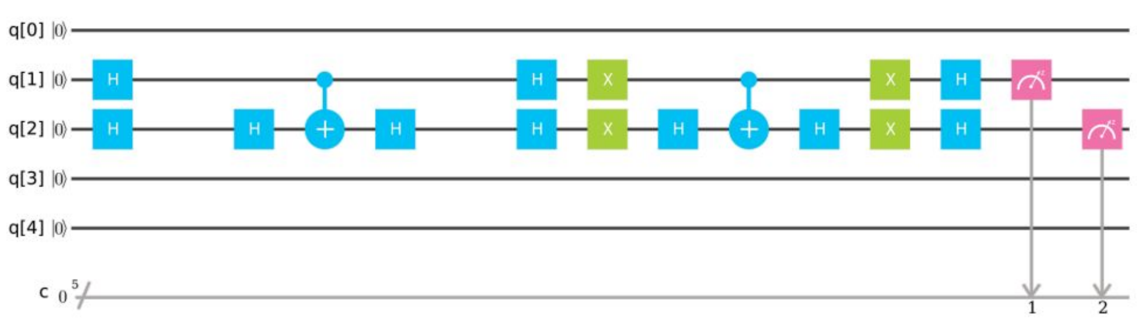
$H_2C_{12}H_2XH|\Psi'\rangle = \frac{1}{2\sqrt{2}}(|1\rangle(|0\rangle - |1\rangle) - (|0\rangle(|0\rangle - |1\rangle) = \frac{1}{2\sqrt{2}}(|10\rangle - |11\rangle - |00\rangle + |01\rangle)$

$XH_2C_{12}H_2XH|\Psi'\rangle = \frac{1}{2\sqrt{2}}(|01\rangle - |00\rangle - |11\rangle + |10\rangle)$

$HXH_2C_{12}H_2XH|\Psi'\rangle = \frac{1}{4}((|0\rangle + |1\rangle)(|0\rangle - |1\rangle) - (|0\rangle + |1\rangle)(|0\rangle + |1\rangle) -$
$(|0\rangle - |1\rangle)(|0\rangle - |1\rangle) + (|0\rangle - |1\rangle)(|0\rangle + |1\rangle))$
$= \frac{1}{4}(|00\rangle - |01\rangle + |10\rangle - |11\rangle - |00\rangle - |01\rangle - |10\rangle - |11\rangle - |00\rangle + |01\rangle + |10\rangle - |11\rangle + |00\rangle + |01\rangle - |10\rangle - |11\rangle$
$= \frac{1}{4}(-4|11\rangle) = -|11\rangle = a$

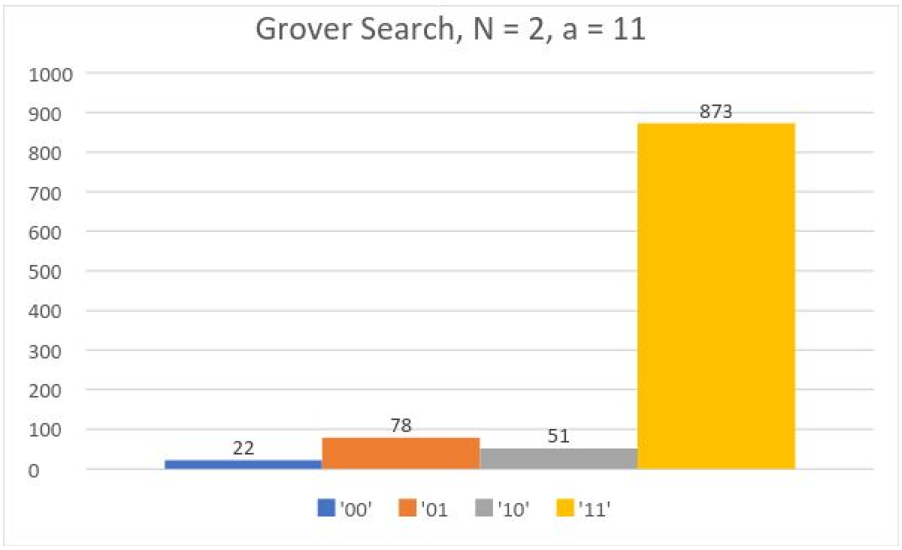
As predicted, the result of a quantum search on $|\Psi'\rangle$ revealed the appropriate $-|11\rangle$ solution, identifying that the element in the 4th position was indeed the element we were looking for. As this solution is probabilistic, it would require multiple iterations to reveal the correct solution on a universal quantum computer. This can be demonstrated when analyzing the results of running Grover's Search Algorithm on a quantum computer and analyzing the results, as will be discussed in the following section.

The quantum circuit for $N = 2$, $a = 11$ is as follows:



The first part of the circuit creates the superposition between the qubits required for the operation. The second section utilizes an oracle U to 'tag' the overall state with the desired outcome. The final part of this circuit performs an oracle U to complete the rotation and reveal the solution. This circuit can be reproduced quickly using the OPENQASM machine code.

The probabilistic results of running this circuit 1024 times utilizing the IBM quantum cloud produces the following results.



As expected, the results of Grover's Search Algorithm produced a probabilistic result that allows us to place a high degree of confidence in the correct answer. This process can be reproduced to provide the solutions to each of the other states $|00\rangle$, $|01\rangle$, $|10\rangle$, however each of these would require a different oracle U to tag the desired result.

While this search algorithm is powerful – producing an O(1) result to a classically O(4) problem – it does have it's limitations. An implicit database where all possible records don't require large amounts of memory consisting of records stored in main memory provides a feasible search structure for a quantum computer. An Explicit database consisting of a distributed network of storage facilities to accommodate a database too large to fit into one computer's memory poses a serious problem. Grover's algorithm requires a quantum superposition of all database records which requires using a superposition of indices of that capacity (i.e. A large number of qubits) as well as localizing classical records in one place, which is impractical for the larges explicit databases.

Quantum Key Distribution

Another powerful promise of quantum computers is secure communications. Quantum Key Distribution (QKD) allows for two parties to utilize the power of quantum mechanics to share a key over a public network while being confident that any potential third party seeking to gain knowledge of the key will be detected. The reason the two parties seeking to share a key can be confident about detecting a third party is due to the unique property of a qubit in a quantum system – any measurement operation disturbs the system and, thus, the key. Once each party can verify that no third party has disrupted the system, the transmitted key can then be used to encrypt the message to be sent over a public channel. If an eavesdropper is detected within the system, the transmission is aborted and another channel can be used.

QKD can be demonstrated on the IBM 5Q quantum computer both using its real computer as well as a simulation. Given the five-qubit limitation of the 5Q, QKD is best demonstrated using a custom typology, allowing us to simulate a greater number of qubits. This algorithm could however be demonstrated using the five qubits of the 5Q, but would only produce a five-digit key which isn't very useful for encrypting/decrypting ASCII based characters. The following is a description of the algorithm for QKD.

Prerequisites:

1. Alice wants to send a message, M consisting of an n length string of bits
2. Alice needs an n bit key, K, to encrypt M such that $M \oplus K = C$, where C is the encrypted message M
3. Bob will also require K to decrypt M such that $C \oplus K = M$, where M is the original n length bit string

Example

Given the message 'Q' is the desired message to send and an arbitrary value K:

$$M = 01010001 \text{, } K = 00110110$$

$$M \oplus K = 01100111 = C$$

$$C \oplus K = 01010001 = M$$

Procedure:

1. Alice generates two random strings of n bits such that $x = x_{n-1}x_{n-2}\ldots x_0$, $y = y_{n-1}y_{n-2}\cdots y_0$
2. Bob generates one random string of n bits such that $z = z_{n-1}z_{n-2}\ldots z_0$
3. Alice prepares each qubit in x, y to produce a partial key k such that:

   $if\ x_i y_i = 00 \rightarrow |k_i\rangle = \quad |0\rangle$

   $if\ x_i y_i = 10 \rightarrow |k_i\rangle = \quad |1\rangle$

   $if\ x_i y_i = 01 \rightarrow |k_i\rangle = \ H|0\rangle$

$$if\ x_i y_i = 11 \rightarrow |k_i\rangle = \ H|1\rangle$$

4. Alice sends her n qubits across public channel to Bob
5. Alice and Bob discard all qubits for which $y_i \neq z_i$
6. The resulting value is the key K which can be used to encrypt/decrypt message M

This procedure can not only be modeled with the 5Q, it can also be implemented easily utilizing IBM's python wrapper which functions completely with the OPENQASM machine code and API. Python's versatility allows us to create a fully functional simulation of QKD. As aforementioned, while this algorithm could be demonstrated using the real quantum computer, the length of n would have to be less than five. As such, the following demonstration of QKD uses a custom typology run on the IBM 5Q. It is written using the python wrapper, however is heavily documented to outline the steps completed. Unnecessary code has been removed to improve readability. The complete, commented, and functional code can be found at the end of this paper.

```
# Alice needs to use an n bit key K to encrypt
# She must first randomly generate two strings of n bits, x & y
# Here, we use n*2 to account for lost information.

x = random_bits(n * 2)
y = random_bits(n * 2)

# Bob generates random 1 string of n bits, z

z = random_bits(n * 2)

# Alice prepares 1 of the following qubits:
#    if xi*yi = 00 -> |xi> =  |0>
#    if xi*yi = 10 -> |xi> =  |1>
#    if xi*yi = 01 -> |xi> = H|0> (+)
#    if xi*yi = 11 -> |xi> = H|1> (-)

k = ''   # Initialize k (partial K)
i = 0    # Initialize index
length = n * 2 – 1    # Initialize length
while length > i:
    if x[i] == '0' and y[i] == '0':
        k += '0'
    elif x[i] == '1' and y[i] == '0':
        k += '1'
    elif x[i] == '0' and y[i] == '1':
        k += '+'
    else:
        k += '-'
    i += 1

# Alice and Bob discard all qubits for which yi != zi

K = ''
i = 0
length = n * 2 - 1
while length > i:
    if y[i] == z[i]:
        K += k[i]
    i += 1
K = K[:8]
```

```python
# If the length of K is less then that of the bits needed to send (n)
# then this loop continues until that requirement is satisfied

# Alice then proceeds to decrypt K herself
#    + -> 0
#    - -> 1

K_Alice = ''
for qubit in K:
    if qubit == '+':
        K_Alice += '0'
    elif qubit == '-':
        K_Alice += '1'
    else:
        K_Alice += qubit

# Alice proceeds to use K (key) to perform gates to each qubit
#    if K[i] = 0 -> do nothing
#    if K[i] = 1 -> x q[i];
#    if K[i] = + -> h q[i];
#    if K[i] = - -> x q[i];
#                -> h q[i];

i = 0
length = n
qasm += '\n// Alice:'
while length > i:
    if K[i] == '1':
        qasm += '\nx q[' + str(i) + '];'      # resulting qasm = nx [qi];
    elif K[i] == '+':
        qasm += '\nh q[' + str(i) + '];'      # resulting qasm = nh [qi];
    elif K[i] == '-':
        qasm += '\nx q[' + str(i) + '];'      # resulting qasm = nx [qi];
        qasm += '\nh q[' + str(i) + '];'      # resulting qasm = nh [qi];
    i += 1

# Bob proceeds to use the same K to perform complementary gates to each qubit
#    if K[i] = 0 -> do nothing
#    if K[i] = 1 -> x q[i];
#    if K[i] = + -> h q[i];
#    if K[i] = - -> x q[i];
#                -> h q[i];
# Bob proceeds to measure each qubit after each operation to uncover K

i = 0
length = n
qasm += '\n// Bob:'
while length > i:
    if K[i] == '+' or K[i] == '-':
        qasm += '\nh q[' + str(i) + '];'      # resulting qasm = nh [qi];
    qasm += '\nmeasure q[' + str(i) + '] -> c[' + str(i) + '];'
    # resulting qasm = measure [qi] -> c[i];
    i += 1

# Initialize API to run state
```

```
api.run_experiment(qasm, device, shots, name, timeout)
```

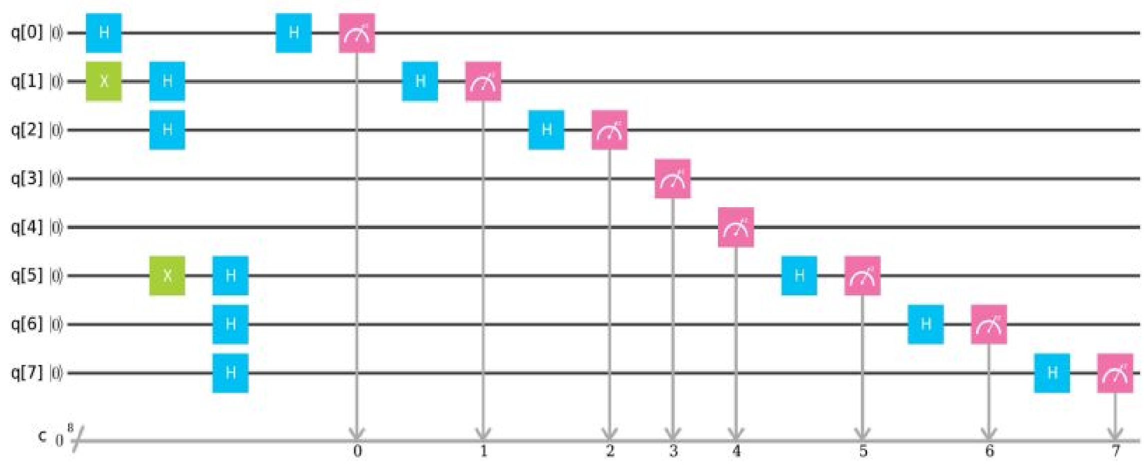
Once the experiment has been run using the *api.run_experiment* method call, that particular experiment can be found in the 'My Scores' section of the IBM Software Development Kit (SDK) online. These results can be analyzed and the value of K can be found. When the code above is run within the main program, the console will output the results of that run. Each run is unique, producing a unique value of K as well as a unique quantum circuit for the QKD. The following is the output describing the operations of Alice and Bob in an example run. Notice that new values of x, y are created until the corresponding length of K after discard is $\geq n$. Also, notice the unique OPENQASM code generated by python to be run on the 5Q.

```
*----- CONSOLE OUTPUT -----*
Message to transmit: Q
Binary message: 01010001
Length: 8
*----- FINDING K >= n -----*
x: 0111010100010100
y: 1000111011111011
z: 0111000001010010
k: +111+-+1+++-+1+
x: 1111001110111001
y: 0101110111011000
z: 0000100000000010
k: 1-1-++1--+1--00
x: 1110011011000011
y: 1011001101110110
z: 0110010001000011
k: -1-+01-+1-++0+-
x: 0100010100001000
y: 1110000111110011
z: 1111010111011000
k: +-+0010-++++10+
K: +-+00-++
*----- QASM -----*
OPENQASM 2.0;

include "qelib1.inc";
qreg q[8];
creg c[8];
// Alice:
h q[0];
x q[1];
h q[1];
h q[2];
x q[5];
h q[5];
h q[6];
h q[7];
```

```
// Bob:
h q[0];
measure q[0] -> c[0];
h q[1];
measure q[1] -> c[1];
h q[2];
measure q[2] -> c[2];
measure q[3] -> c[3];
measure q[4] -> c[4];
h q[5];
measure q[5] -> c[5];
h q[6];
measure q[6] -> c[6];
h q[7];
measure q[7] -> c[7];
Execution ID:
c6b41b0cd3000289aa10f724e74929e6
*----- RESULTS -----*
{'measure': {'labels': ['00100010'],
'qubits': [0, 1, 2, 3, 4, 5, 6, 7],
'values': [1]}}
Alice's key: 01000100
Bob's key: 01000100
*----- ENCRYPTION -----*
Original ASCII message: Q
Original binary: 01010001
Encrypted binary using Alice's key:
00010101
Decrypted binary using Bob's key:
01010001
Resulting ASCII message: Q
```

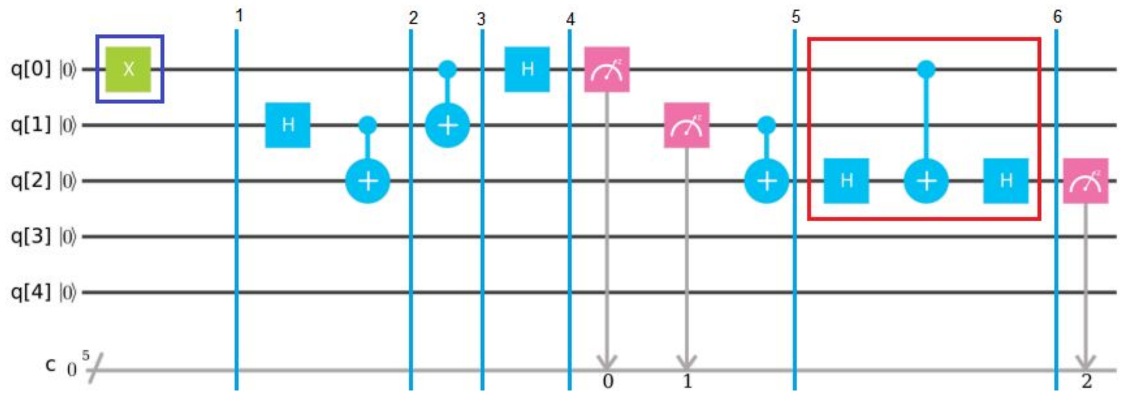
The following is the quantum circuit produced by the algorithm on this one-time run, displaying exactly the OPENQASM machine code generated via the Python wrapper:



As can be observed in the program, this algorithm leaves no room for Eve to measure any part of the system prior to Bob's measurement without somehow disrupting the output key. As such, this algorithm provides a tremendous advantage over traditional communications and, if combined with the one-time pad algorithm, can be near unbreakable. This advantage is one of the greatest drivers of investment pushing the development of a universal 50+ qubit quantum computer for business and government applications.

Teleportation

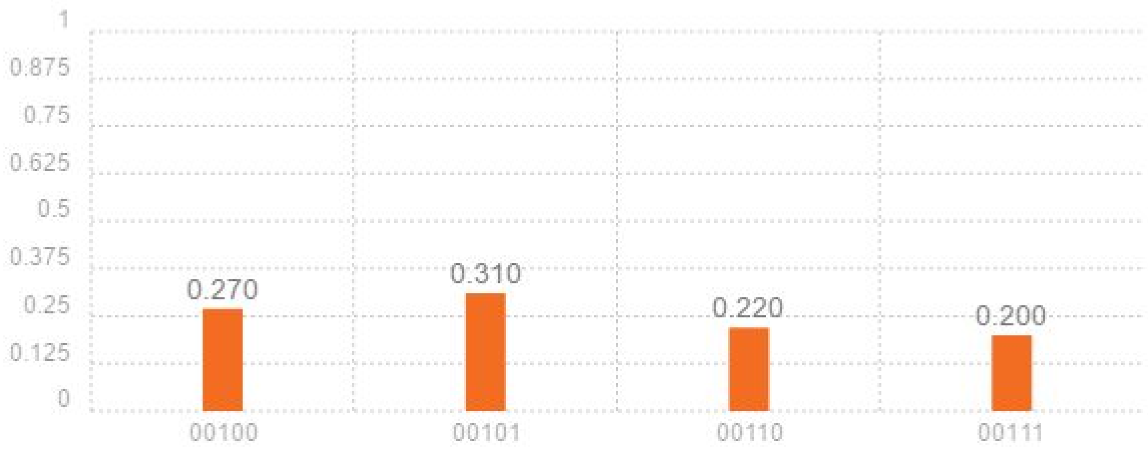
Although we were not able to test teleportation at a distance, we were able to run the teleportation algorithm (with slight modification) on the 5Q to prove that it works.



Q[0] is the qubit, $|\Psi\rangle$ we want to "teleport".

The gates in the red rectangle perform a controlled phase flip (CZ) on q[2], with q[0] as the control. This was necessary as the computer does not have a CZ gate. The gate in the blue rectangle is used for state preparation, for this example we are just sending a $|1\rangle$.

Below are the results from the simulator:



I opted to include simulator results first to eliminate quantum errors. The register reads from right to left, so q[0] is the rightmost bit, and the middle bit is always 1, which is the output that was teleported. This shows that for a given input, the values Alice will measure can vary but the output will still be correct as long as bob applies the x and z gates correctly (or doesn't apply them, if he is not supposed to).

Steps:

Step 1: start with a qbit, $|\Psi\rangle$, in $q[0]$ (in this case though we just let $|\Psi\rangle = |1\rangle$)

$|q[2]q[1]q[0]\rangle = |001\rangle$

Step 2: entangle $q[1]$ and $q[2]$:

$|q[2]q[1]q[0]\rangle = \frac{1}{\sqrt{2}}(|001\rangle + |111\rangle)$

Step 3: Perform a controlled not on $q[1]$ with $q[0]$ as control

$|q[2]q[1]q[0]\rangle = \frac{1}{\sqrt{2}}(|011\rangle + |101\rangle)$

Step 4: Perform a Hadamard gate on $q[0]$

$|q[2]q[1]q[0]\rangle = \frac{1}{\sqrt{2}}(|01\rangle H|1\rangle + |10\rangle H|1\rangle) = \frac{1}{2}(|010\rangle - |011\rangle + |100\rangle - |101\rangle)$

Step 5: apply a CNOT gate to $q[2]$ with $q[1]$ as control (or, Alice measures $q[1]$ and if it is 1, Bob applies X on $q[2]$)

$|q[2]q[1]q[0]\rangle = \frac{1}{2}(|110\rangle - |111\rangle + |100\rangle - |101\rangle)$

Step 6: apply CZ on $q[2]$ with $q[0]$ as control (or, Alice measures $q[0]$, and if it is 1, Bob applies Z on $q[2]$)

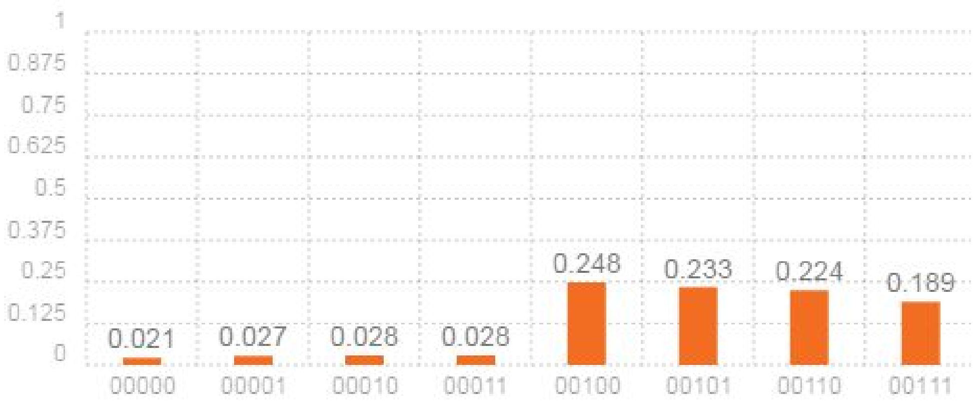
$|q[2]q[1]q[0]\rangle = \frac{1}{2}(|110\rangle + |111\rangle + |100\rangle + |101\rangle)$

(Note: in the section called "Challenges with the 5Q", we proved the sequence of gates used in the red box were equivalent to CZ

So,

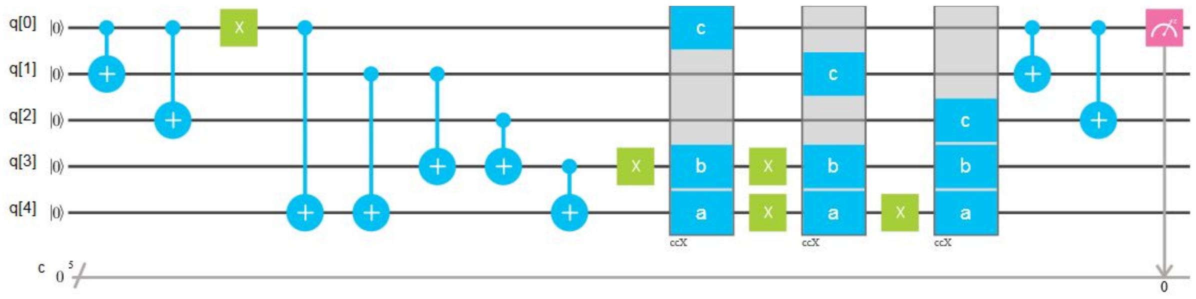
$|q[2]\rangle = |1\rangle$, which is what we wanted to teleport.

Below are the results from the 5Q, as you can see the correct answers are most common, but due to quantum errors the incorrect answers do still occur:

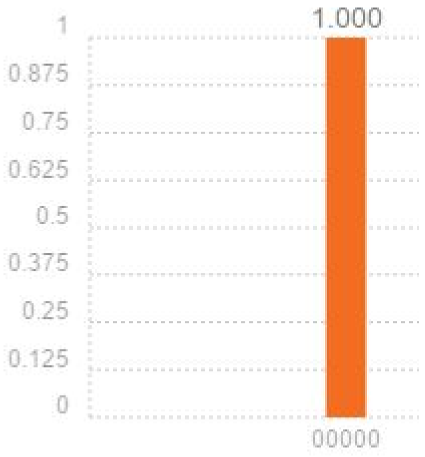


Simulated Quantum Error Correction

This algorithm takes 3 qubits as input (q[0]-q[2] in this case), and corrects bit flip errors. The algorithm is capable of correcting 1 error in the 3 qubits correctly. Below is the full circuit, and q[0] has been deliberately flipped to demonstrate how it works.



The output, q[0] at the end, is still 0:



Unfortunately, due to the use of CCNOT gates, we were not able to run this on the 5Q. Fortunately, the IBM Quantum Experience allows users to define custom topologies for simulation, which allow use of additional gates such as CCNOT. In custom topologies, all qubits are connected to all other qubits for control, so it is easier to design circuits.

Deutsch-Jozsa Algorithm

The Deutsch-Jozsa algorithm, proposed by David Deutsch and Richard Jozsa in 1992, was one of the first examples of a quantum algorithm to show a separation between the efficiency of a classical and quantum algorithm. Consider a function $f(x)$ that takes as input n-bit strings and returns 0 or 1. The function is known to be either *constant* or *balanced*. The function is called *constant* if the function is the same on all inputs (i.e. $f(x) = 0,\ \forall x$ or $f(x) = 1,\ \forall x$). The function is called *balanced* if the function is 0 on half of the possible inputs and 1 on the other half of possible inputs (i.e. $f(x) = 0,\ \forall x \in S$ and $f(x) = 1,\ \forall x \neq S$ and $|S| = 2^{n-1}$). A conventional classical algorithm requires $2^{n-1} + 1$ in the worst case to distinguish whether the function is constant or balanced. Using this quantum algorithm, only one function evaluation is sufficient.

The following steps are required to implement the Deutsch-Jozsa algorithm:

1. Initialize n qubits in the all-zeros state $|0, \ldots, 0\rangle$
2. Apply the Hadamard gate to each qubit
3. Apply the oracle circuit U_f (i.e. the quantum circuit that implements f)
4. Repeat step 2
5. Measure each qubit. Let $y = (y_1, \ldots, y_n)$ be the list of measurement outcomes

If f is a constant function, then y will be the all-zeros string. Otherwise, f is a balanced function. From a mathematical point of view, consider the input state $|\psi_0\rangle = |0\rangle^{\otimes n}|1\rangle$ that describes the n qubits $|x\rangle$ in the all-zeros state and a register for the function output, initialized to the 1 state. After applying the Hadamard gates to each qubit, the state of the registers is now

$$|\psi_1\rangle = \sum_{x \in \{0,1\}^n} \frac{|x\rangle}{\sqrt{2^n}} \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right]$$

Applying the oracle circuit U_f transforms the state $|x, y\rangle \longrightarrow |x, y \oplus f(x)\rangle = (-1)^{f(x)}|x, y\rangle$.

$$|\psi_2\rangle = U_f|\psi_1\rangle = \sum_x \frac{(-1)^{f(x)}|x\rangle}{\sqrt{2^n}} \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right]$$

After discarding the last qubit and applying the final Hadamard gates, the resulting state becomes

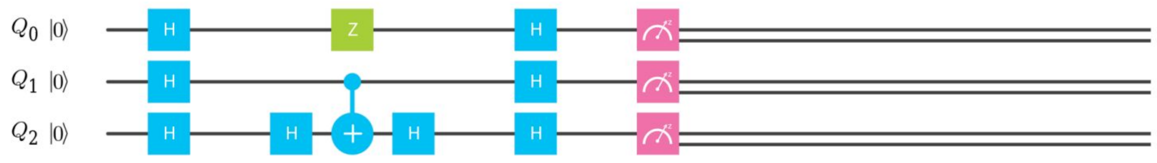
$$|\psi_3\rangle = \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} (-1)^{f(x)} \left(\frac{1}{\sqrt{2^n}} \sum_{z \in \{0,1\}^n} (-1)^{x \cdot z} |z\rangle\right) = \sum_{z \in \{0,1\}^n} \left(\frac{1}{2^n} \sum_{x \in \{0,1\}^n} (-1)^{f(x) + x \cdot z}\right) |z\rangle$$

The amplitude associated with the classical state $|0\rangle^{\otimes n}$ is

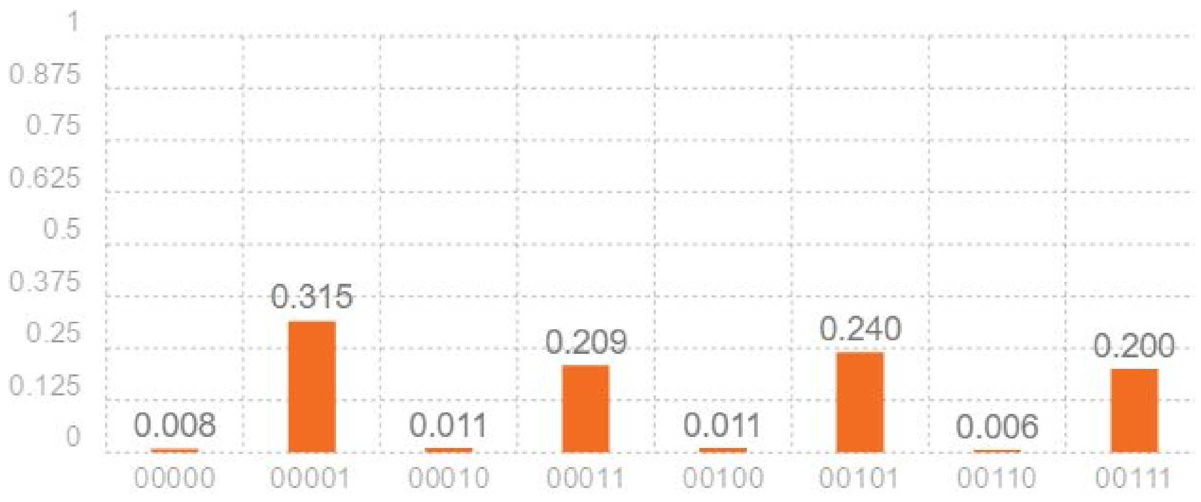
$$\frac{1}{2^n} \sum_{x \in \{0,1\}^n} (-1)^{f(x)}$$

In the case where f is constant, the amplitude associated with $|0\rangle^{\otimes n}$ is $+1$ or -1 depending on the value $f(x)$ takes. However, when $f(x)$ is balanced, the amplitude will be 0 since all terms in the sum over x cancel each other so a measurement must yield a result other than 0 on at least one qubit in y.

Consider the following example where $n = 3$ and $f(x) = x_0 \oplus x_1 \oplus x_2$. This balanced function can be implemented with the oracle circuit $U_f = Z_0 C Z_{12}$ such that $Z_0|x\rangle = (-1)^{x_0}$ and $CZ_{12} = (-1)^{x_1 x_2}|x\rangle$.



The results from running this circuit on the IBM 5Q are shown below.

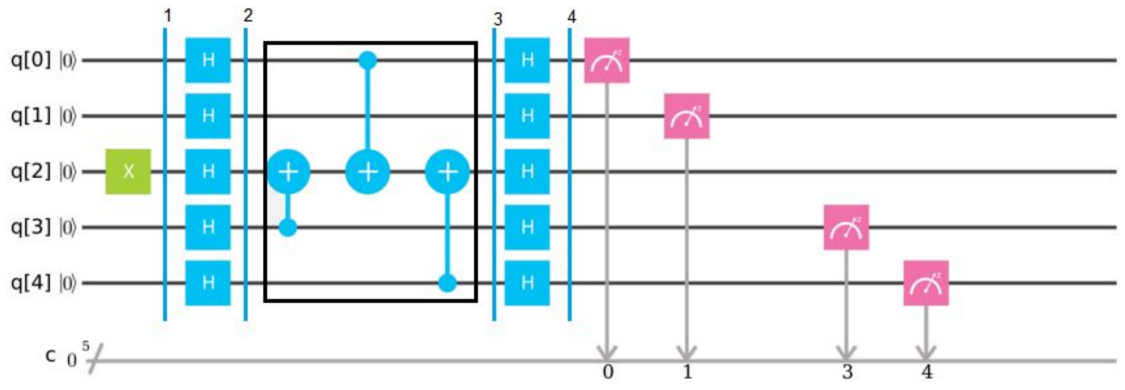


Since the all-zeros string occurs with almost 0 probability, this shows that the function is balanced. Removing the gates associated with U_f results in f being constant. With this change, the probability of observing the all-zeros string becomes $0.971 \approx 1$ as expected.

Bernstein - Vazirani Algorithm

The Bernstein - Vazirani algorithm uses property $CNOT_{12} = H_1 H_2 CNOT_{21} H_1 H_2$ to determine in a single query the value of a in $f(x) = a \bullet x \ (bitwise\ mod\ 2)$. Such a function is represented as a series of CNOT gates - from each input register where a is 1 - on the output register, q[2]. A classical computer would need to make n queries to determine a, where n is the number of bits in a.

Below is the circuit for the B-V algorithm with a = 1011, as done in class. The gates in the black box are supposed to be hidden, and this circuit demonstrates that their value can be determined in a single query by the B-V algorithm. In this circuit, q[2] is the output and the other qubits are the input.



With the current example, these are the steps:
Step 1: Apply x to the output (q[2])

$q[0] = q[1] = q[3] = q[4] = |0\rangle$

$q[2] = |1\rangle$.

Step 2: Apply Hadamard gates to every qubit

$q[0] = q[1] = q[3] = q[4] = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$

$q[2] = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$

Step 3: Make single query, U, on the qubits:

For $x \varepsilon \{0, 3, 4\}$ (qubits in U which are used as control),

$|q[x]q[2]\rangle = CNOT(\frac{1}{2}(|00\rangle + |01\rangle + |10\rangle - |11\rangle)) = \frac{1}{2}(|00\rangle + |01\rangle + |11\rangle - |10\rangle)$, so,

$q[0] = q[3] = q[4] = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$,

$q[1] = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$, unchanged from step 2, and

$q[2] = \frac{1}{\sqrt{2}}(|1\rangle - |0\rangle)$.

Step 4: Apply Hadamard gates to each qubit again

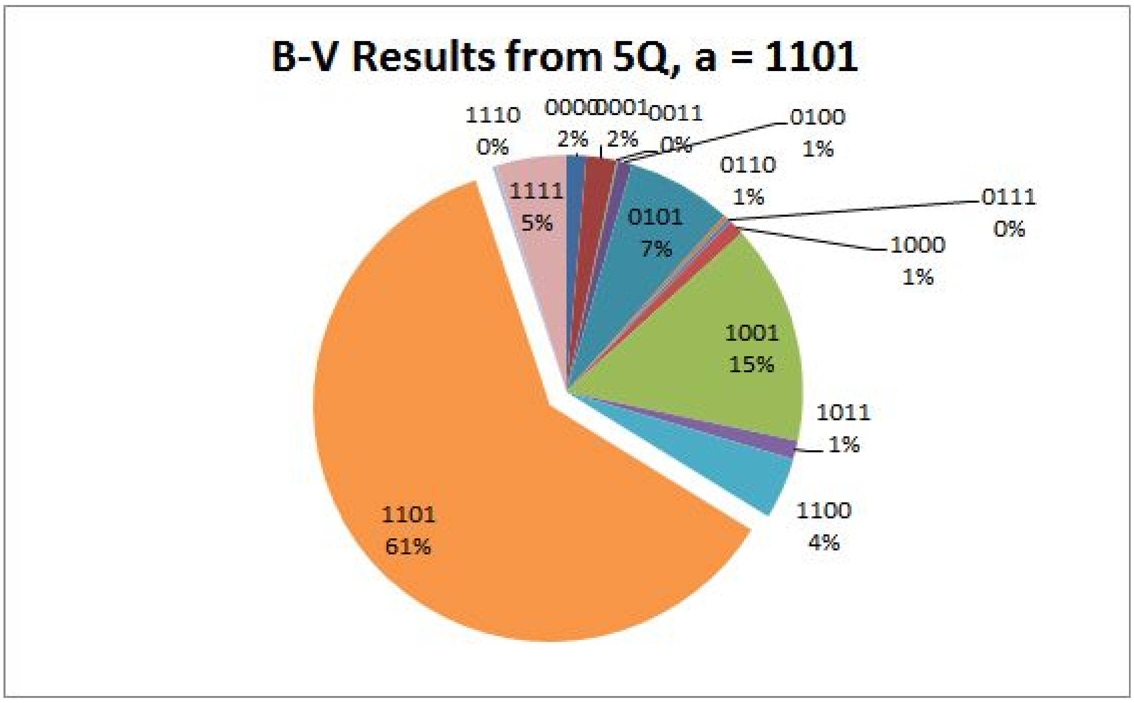
$q[0] = q[3] = q[4] = |1\rangle$

$q[1] = |0\rangle$

$q[2] = -|1\rangle$

Now, the input qubits ($q[4], q[3], q[1], q[0]$) are measured, and the result is 1101, which is a.

Below is a graph pie graph showing the percentage of the time we got each outcome on the 5Q computer, with 1024 shots.



The correct outcome, 1101, comes out most often at 61%. If you run the algorithm in the simulator, you always get 1101 as the outcome. (Note: normally the output would consist of 5 bits, but I removed the middle bit, corresponding to q[2], since it was never measured and isn't relevant)

Conclusion

The IBM 5Q coupled with the Quantum Experience API has been laying the foundation for the development of future technologies. Natively, it has paved the road for the IBM Q which is poised to offer a ~50 qubit computer, capable of much better functionality with respect to available gates and connections between qubits. These challenges with the 5Q limit the algorithms that can be implemented on it's hardware. The IBM Q, on the other hand, will not have these limitations. As this paper has displayed, there are still numerous algorithms and experiments that can be run on the 5Q as well as its simulator that are functional and prove to be reliable. While these experiments are only elementary, they are still an exciting demonstration of the potential of a real, universal quantum computer.

Supporting Documentation

Please find attached.