IBM Quantum Experience Review

Boettner, Gopalan, Neumann

May 2020

Figure 1: The IBM Q System One Quantum Computer

1 The IBM Q System One

All computing systems rely on a fundamental ability to store and manipulate information. Classical computers manipulate individual bits, which store information as binary 0 and 1 states. Quantum computers instead leverage quantum mechanical phenomena to manipulate information. To do this, they rely on quantum bits, or qubits, which are unit vectors in $\mathbb{C}^2$. Storing and manipulating data in the form of qubits leads to more information. This opens up the possibilities for faster and more efficient algorithms that allow quantum computers to tackle problems that would be harder or practically impossible to solve on a classical computer.

The history of quantum computing research goes all the way back to 1927 and each result marked a new piece of the puzzle. In October 2017, a significant mathematical result was published in the paper "Quantum Advantage with Shallow Circuits" (Bravyi et al. 2018), which guided the development of algorithms for quantum computing. The significance of this algorithm was that unlike Shor's algorithm, it proved that a quantum computer can solve certain problems with near certainty in a fixed number of steps, no matter the increased input. A classical computer, on the other hand, would require an increased number of steps as the input increases for the same problems. This made the Quantum Advantage algorithm a solid and necessary brick in the foundation of quantum computing.

Building on this foundation, in 2019 IBM Quantum designed and built the world's first integrated quantum computing system for commercial and educational use: IBM Q System One[1]Å large goal of this project was to further quantum computing by providing a reliable resource for quantum research. IBM Q System One provides a superconducting quantum computer that operates outside of research labs allowing the general public to access it for free. In this paper, we describe using the IBM software, recreate circuits to experiment with the Grover's Search algorithm and the Deutsch-Josza algorithm, and describe our results.

[1]https://thequantumdaily.com/2020/01/30/getting-real-time-information-on-ibm-quantum-devices/

2 Coding in IBM Q

IBM Q was designed to make computing the outcome of Quantum circuits as simple as possible. The built in Circuit Composer allows users to intuitively click and drag universal quantum gates onto a set of qubits. The user can then view the **QASM** language[2] encoding of their circuit and manipulate it further if they wish. Finally, IBM's cloud based compute model allows the user to run their encoding on one of IBM's servers where it will join a queue before the results of the quantum circuit are sent back to the user. This whole process is highly documented and user friendly. This section will outline how to use the Circuit Composer in greater detail as well as provide some simple examples of the interface.

2.1 Circuit Composer

The IBM Q Circuit Composer presents the circuit visual as well as the **QASM** circuit encoding side by side. By allowing user interaction with either, this platform helps the user quickly learn how to tweak their circuit exactly as they want it. This visual interface provides icons for each gate along with a glossary entry for each icon that explains what the gate does. Despite providing the option to write code, the Circuit Composer in no way forces the user to have past programming experience. The QASM representation simply allows more precise control over the circuit creation process.

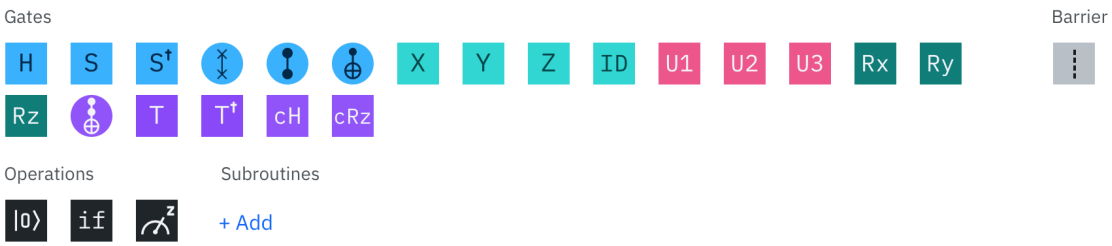

Figure 2: Gates in the Circuit Composer

[2]https://github.com/Qiskit/openqasm

2.2 Encoding

Circuit encoding in IBM Q is simple. The software uses the QASM language developed by IBM in 2017 and uses arrays to represent qubit input states. The QASM provides a set of quantum gates that are represented with letters and take qubits as inputs. The order in which these gates appear in the code correlates to the order that they appear on the circuit. This encoding can then be interpreted by the IBM servers in order to compute the results of the circuit.

Encoding each circuit into a small file is a crucial element of IBM Q software for a number of reasons. Most importantly, the simplified and optimized encoding helps combat the severe limitations of our current quantum computing capabilities. The optimization occurs when IBM transpiles[3] the circuit encoding. In this process, the number of quantum gates is reduced as much as possible while maintaining the circuit's properties. Additionally, the small file size minimizes data transfer between users and IBM servers which allows faster and more convenient access for everyone.

2.3 Getting Results

As previously mentioned, IBM Q is fully cloud based. This system utilizes multiple global servers to process user circuits. When a user submits their circuit to be run a specified number of times, it joins a server queue. These servers utilize classical computers to decode the quantum circuit and send the corresponding qubits to a real quantum computer. The quantum computer then runs the circuit as many times as specified and returns the results back to the user. The results use a histogram to display each returned state along with probability with which that state appeared. These results also include the time it took for the circuit to transpile, validate, queue, and run.

[3]The transpiler introduces the concept of a pass manager to allow users to explore optimization and find better quantum circuits for their given algorithm (Qiskit 20).

3 Theory

3.1 Deutsch-Jozsa algorithm

The Deutsch-Jozsa algorithm was one of the first quantum algorithms to show a separation between the quantum and classical difficulty of a problem. Formally, it is defined as follows: Consider a function $f(x)$ that takes as input n-bit strings x and returns 0 or 1. We are promised that $f(x)$ is either a *constant* function that takes the same value $c \in 0, 1$ on all inputs x, or a *balanced* function that takes each value 0 and 1 on exactly half of the inputs. **The goal is to decide whether f is constant or balanced by making as few function evaluations as possible**.

Our class performed a simple implementation of the Deutsch-Jozsa algorithm in which we used the 'phase kickback trick' for an oracle to guess the sum of two bits using only a single query. Classically this would take 2 queries. In our tests with the IBM software, we implement a more 'quantum-like' experimental setup that is described **here**[4]. In this setup, consider a typical interference experiment where a particle that behaves like a wave, such as a photon, can travel from the source to an array of detectors by following two or more paths simultaneously. The probability of observing the particle will be concentrated at those detectors where most of the incoming waves arrive with the same phase. We can set up the 2^n possible paths and 2^n detectors with n-bit strings x and y where x denotes a path and y denotes a detector. Though this experiment is not practical since it would require an impossibly large optical table, we can simulate this experiment on a quantum computer with just n qubits and access to the oracle circuit U_f. Classically, this method requires $2^{n-1} + 1$ function evaluations in the worst case, but using the Deutsch-Jozsa algorithm the question can be answered with just one function evaluation.

Suppose that the phase accumulated along a path x to a detector y equals $C(-1)^{f(x)+x.y}$ where $x.y = \sum_{i=1}^{n} x_i y_i$ is the binary inner product and C is a normalizing coefficient. The probability to observe the particle at a detector y can be computed by summing up amplitudes of all paths x arriving at y and taking the absolute value squared,

[4] see https://quantum-computing.ibm.com/docs/guide/q-algos/deutsch-jozsa-algorithm

i.e.,:

$$Pr(y) =\mid C \sum_{x} (-1)^{f(x)+x.y} \mid^2 \tag{1}$$

The normalization condition, $\sum_y Pr(y) = 1$, gives us $C = 2^{-n}$. Now, we can compute the probability of observing the particle at the detector $y = 0^n$ (the all zeros string). We have $Pr(y = 0^n) =\mid 2^{-n} \sum_x (-1)^{f(x)} \mid^2$. If $f(x) = c$ is a constant function, we get $Pr(y = 0^n) =\mid (-1)^c \mid^2 = 1$. However, if $f(x)$ is a balanced function, we get $Pr(y = 0^n) = 0$, since all terms in the sum cancel each other out. We can therefore determine whether f is constant or balanced with certainty by running the experiment just once. The Deutch-Josza algorithm is as follows:

1. Initialize n qubits in the all-zeros state $|0, 0, \cdots, 0\rangle$.
2. Apply the Hadamard gate H to each qubit.
3. Apply the oracle U_f.
4. Repeat Step 2.
5. Measure each qubit. Let $y = (y_1, \cdots, y_n)$ be the list of measurement outcomes. We find that f is a constant function if y is the all-zeros string.

The applications of Hadamards in Step 4 maps a basis state to a superposition $2^{-n/2} \sum_y (-1)^{x.y} |y\rangle$. Thus the state reached after Step 4 is $|\Psi\rangle = \sum_y \Psi(y) |y\rangle$, where $\Psi(y) = 2^{-n} \sum_x (-1)^{f(x)+x.y}$. This is exactly what we need for the interference experiment described above. The final measurement at Step 5 represents detecting the particle. The probability to measure in Step 5 is 1 if f is a constant function and 0 if f is a balanced function. Thus the Deutsch-Jozsa problem is solved with certainty by making just one function evaluation.

3.2 Grover Search algorithm

One major benefit of Quantum computers is the ability to search databases much faster than classical computers. The Grover's search algorithm is a great example of how quantum computers obtain this advantage. When applied in an unstructured

search, Grover's Search algorithm can speed up the problem quadratically. The experiments described in this paper are modeled on the resource notes provided **here**[5].

In essence, an unstructured search is the task of finding a desired item within an unordered list of N items. In classical computing, this search takes N steps in the worst case scenario and $N/2$ steps on average. A quantum computer using Grover's Search algorithm can reduced this search to $\sqrt{N}$ steps. An additional benefit of Grover's search algorithm is that it does not require the list to have a specific internal structure. This makes the algorithm a general solution for a wide variety of database search problems.

Consider an unstructured list of N items. Each item is denoted x except the item we are searching for which will be denoted w. The tactic used to search this unstructured list exploits Grover's amplitude amplification trick by repeatedly applying an oracle U_f and rotating about a superposition state. This oracle encodes all items within the list as a function f which returns $f(x) = 0$ for all $x \neq w$ and $f(x) = 1$ for $x = w$. This function is encoded as a unitary matrix which allows a quantum computer to interpret each item as a superposition to the function. Here, for the basis states $|x\rangle$, the U_f unitary matrix is given as $U_f |x\rangle = (-1)^{f(x)} |x\rangle$.

Since w could be anywhere within the list, the general strategy is to try the uniform superposition state as input for the oracle:

$$|s\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle$$

When we measure this state, we measure all states with equal probability $(1/N)$. This is where amplitude amplification comes in. The algorithm utilizes geometrical interpretations of two reflections, which generate a rotation in a two-dimensional plane. The only two special states we need to consider are the desired item $|w\rangle$, and the uniform superposition $|s\rangle$. Though these two vectors span a two-dimensional plane in the vector space $\mathbb{C}^2$, they are not quite perpendicular. Thus we introduce an additional state $|s'\rangle$ that represents the span of these two vectors, which is per-

[5]see https://quantum-computing.ibm.com/docs/guide/q-algos/grover-s-algorithm

pendicular to $|w\rangle$ and is obtained from $|s\rangle$ by removing $|w\rangle$ and re-scaling. The Grover's Search algorithm is as follows:

1. Start with the uniform superposition state $|s\rangle$. $|\psi_0\rangle = |s\rangle$
2. Apply the oracle U_f to the state $U_f |\psi_t\rangle = |\psi_t'\rangle$
3. Apply another rotation U_s about $|s\rangle$ using $U_s = 2|s\rangle\langle s| - 1$, giving us $|\psi_{t+1}\rangle = U_s U_f |\psi_t\rangle$
4. Repeat step 2.

The two reflections result in a rotation. The transformation $U_s U_f$ rotates the initial state $|s\rangle$ closer towards the desired item $|w\rangle$. Since the average amplitude has been lowered by the first reflection, this transformation boosts the negative amplitude of $|w\rangle$ to roughly three times its original value, while it decreases the other amplitudes. Step 2 is then repeated several times which allows the algorithm to get closer and closer to the desired item $|w\rangle$. After t steps we have $|\psi_t\rangle = U_s U_f |\psi_0\rangle$. After roughly $\sqrt{N}$ rotations, the algorithm will zero in on w, as we can see that the amplitude of $|w\rangle$ grows linearly with the number of applications $\approx tN^{-1/2}$.

4 Expressing Circuits in IBM Q

For both Deutsch-Jozsa and Grover's Search algorithms, IBM had included numerous examples of their equivalent circuits. We decided to test these examples to see if they held up to their theoretical results. Using the given examples allowed us to test IBM's implementations and makes sure there were no errors in our own encoding. We verified that these encodings matched the algorithms from our notes and then ran them multiple times through the IBM servers. The subsections bellow detail our results.

4.1 Deutsch-Jozsa algorithm

For this example we tested a constant and balanced function with the Deutsch-Jozsa algorithm. IBM's example for a balanced function over three qubits uses Hadamard gates, a C-not and a controlled Z-gate. This function should never return the all

zero state. In contrast, the constant function example simply Hadamards each qubit twice with no function in the middle (this is clearly a constant function as it is the Deutsch-Jozsa of the trivially constant function). When the first example circuit is run through the IBM Q software, it returned even results for each state but the all zero qubit state. This suggests that the function is balanced since the Deutsch-Jozsa algorithm never returns the string of all zeros. In contrast, for the trivial constant function the result was always the state of all zero qubits. This clearly shows that the software correctly evaluated both example circuits. We ran these simulations multiple times without error or false positives. This demonstrates that the quantum computer can reliably evaluates these circuits.

```
4  qreg q[5];
5  creg c[5];
6  h q[1];
7  cx q[2],q[3];
8  s q[1];
9  h q[2];
10 h q[2];
11 cx q[1],q[2];
12 s q[1];
13 h q[2];
14 h q[1];
15 h q[2];
16 x q[1];
17 x q[2];
18 h q[2];
19 cx q[1],q[2];
20 x q[1];
21 h q[2];
22 h q[1];
23 x q[2];
24 h q[2];
25 measure q[1] -> c[1];
26 measure q[2] -> c[2];
```

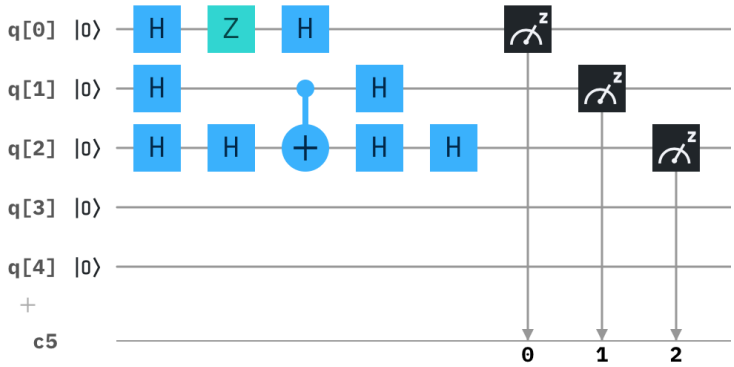

Figure 3: Deutsch-Jozsa on Balanced Function

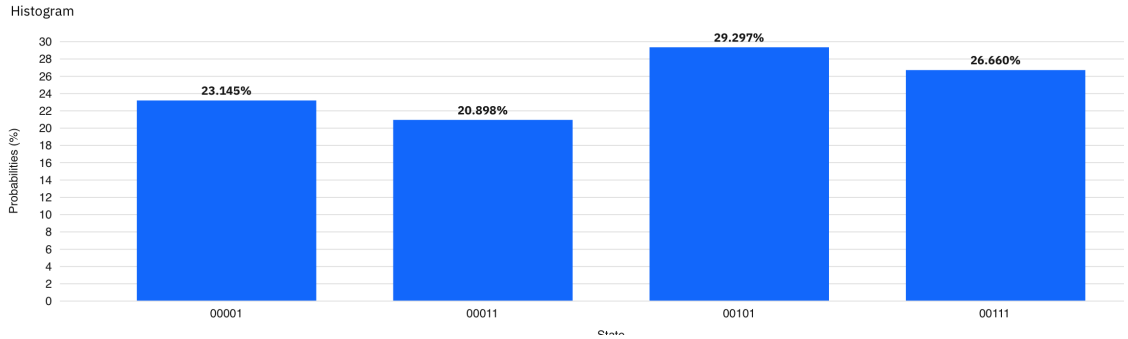

Figure 4: Result for Balanced Function

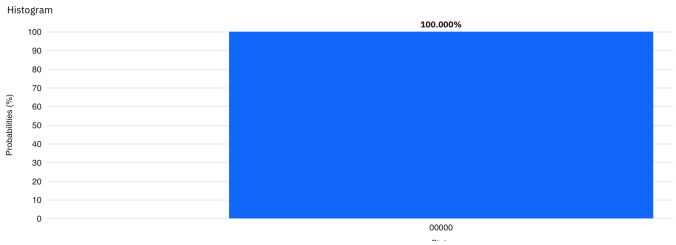

Figure 5: Result for Constant Function

4.2 Grover Search algorithm

The Grover search algorithm finds a given element inside of a list or database. As mentioned in the theory section, the number of necessary oracles for an n-qubit state is $\sqrt{n}$. For the example IBM provided, there are two qubits and four necessary oracle circuits. Each of these circuits corresponds to a given output state and if the quantum computer works properly, they would each only return one result with high probability. In this case IBM Q interpreted the four oracle circuits correctly and each only returned the expected state. This suggests that the software properly evaluated each circuit and that their example of the Grover search algorithm matched the one we studied in class.

```
qreg q[5];
creg c[5];
h q[1];
cx q[2],q[3];
s q[1];
h q[2];
h q[2];
cx q[1],q[2];
s q[1];
h q[2];
h q[1];
h q[2];
x q[1];
x q[2];
h q[2];
cx q[1],q[2];
x q[1];
h q[2];
h q[1];
x q[2];
h q[2];
measure q[1] -> c[1];
measure q[2] -> c[2];
```

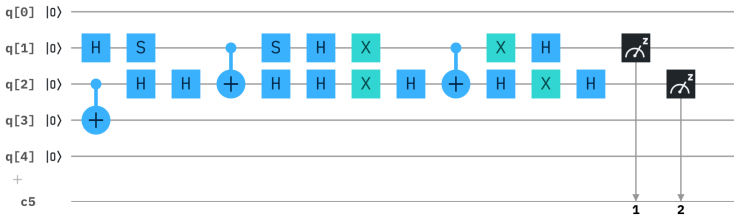

Figure 6: Grover Search algorithm (a)

5 Conclusion

There are still many obstacles to overcome before achieving general purpose quantum computing. In order to distinguish between states and make any sort of calculation, a two level system is required. While bits in classical computing can be easily flipped and distinguished, physical qubits are extremely volatile and will rapidly decay their energy into the environment. The current most promising method of maintaining a two level system with superconducting qubits is to keep them at extremely low temperatures. For example, the IBM Quantum Experience keeps its superconducting qubits at 15 milliKelvin in order to reduce ambient heat and noise

from interfering with the system. Despite this, our current technology and understanding only allows qubits to be maintained at an excited state for a small number of microseconds. Thus any quantum computation that takes longer than a small number of microseconds will become meaningless. This severely limits the current potential of quantum computers in tackling practical problems.

Due to this severe limitation, quantum computing does not currently have many general purpose abilities. While the IBM quantum computer is fascinating to experiment with and prove various results, its computational power is quite limited. The IBM Q software was designed to allow academics and engineers to test circuits on an actual quantum computer and further software development with quantum algorithms. While the current IBM quantum computer does not provide practical benefits, it still benefits our understanding of quantum computation for when quantum computers become more accessible. IBM's software is easy to approach and use while still allowing for more complicated research and application. This paper only skims the surface of IBM's potential. Further research and testing will reveal the full capability of IBM's quantum software.

6 Appendix

6.1 Qiskit

Qiskit is a framework developed around QASM to make it possible to work with IBM Q from a console. For the purposes of this project we didn't create any Qiskit files, but it would be very useful for future research. This framework can be installed as a Python library through the pip package manager. When a user inputs their API key for the IBM Q software they can send any QASM circuit to one of the backend servers for processing. The server responds with a dictionary of each possible qubit state and the number of runs that resulted in that state. This interface allows for more nuanced work and allows a user to save configuration and circuit settings between queries. For the average user, Qiskit adds a lot of unnecessary complexity and code to the process of drawing and testing a quantum circuit. For researchers and engineers however, it allows far more flexible configurations and setups for testing quantum circuitry. We would highly recommend the Qiskit framework for work utilizing the IBM Q quantum computing platform.