Running Algorithms on Google and IBM Quantum Computers

Joshua Vendrow and Zack Berger

March 21, 2021

1 Introduction

We previously implemented six algorithms using Cirq and ran them on the Cirq simulator. These included Deutsch-Jozsa, Bernstein-Vazirani, Simon, Grover, Shor, and QAOA. In this project, we ran our implementations on Google Sycamore and a variety of IBM computers, and added basic error correction. In order to run on IBM computers, we ported our Cirq implementation to Qiskit. To convert from Cirq to Qiskit, we bridged via the .qasm file extension:

```
// Porting Cirq to Qiskit
cirq_circuit = make_circuit()
qiskit_circuit = qiskit.QuantumCircuit.from_qasm_str(cirq_circuit.to_qasm())
```

In Section 2.1, we describe out testing procedure and visualize the resulting accuracies and run times for running our implementations of the quantum circuits on a variety of simulators and quantum computers. At the end of this section, in Subsection 2.8, we example scalability in n and compare the results on the computers. In Section 3, we describe out attempts at adding error correction to the circuits and give initial results and analysis. Finally, in Section 4 we describe the modifications made to our programs in order to move then from the simulators to the quantum computers.

2 Program Testing and Results

In this section we describe our efforts to test our implementations of the Quantum algorithms and report accuracies for each circuit on the simulators and computers. In Section 2.1 We provide an overview and motivation for our testing efforts, and present specific details about the tests for each algorithm in their corresponding subsections. We also report a limit number of results for execution time on IBM computers.

2.1 Testing

For each of the six algorithms, we create a series of tests at each possible number of qubits, and run each test case on a range of simulators of quantum computers. For Deutsch-Jozsa, Bernstein-Vazirani, Simon, and QAOA, we automatically generate a random set of tests at each possible number of qubit. We generate up to 10 tests for every number of qubits, with a lower number of tests at smaller numbers of qubit based on the number of possible unique tests. For Grover and Shor's we only create a small number of set experiments due to limitations in the implementation.

For each algorthm, we measure the accuracy of the results produced by a set of computers and compare these results to Cirq and Qiskit simulators are baselines for the accuracy. Thus, any incorrect behavior worse than that of the simulators can likely be attributed to error on the computer itself.

In order to be able to report accuracy results, we run our circuits on each simulator and computer for at least 1024 repetitions. In most cases we average these results, but in some cases in which the data is meant to be read in as a stream of result (Eg. in Simon's algorithm) we examine our repetitions in a random order.

We limited the number of quantum computers we ran in order to be able to report and evaluate a reasonable number of experiments, but we decided to run all seven available quantum computers on Deutsch-Jozsa in order to compare the run time and accuracy of all computers. We were also unable to run Grover's algorithm and Shor's algorithm on Sycamore becuase of limits in the gate set (e.g. for Grover we needed to use a CCNOT gate which Sycamore does not allow).

2.2 Deutsch-Jozsa

For Deutsch-Jozsa, we ran experiments on a small number of constant functions and a large number of balanced function; our possible number of unique experiments was limit because there are only two possible balanced functions at each number of qubits n. We randomly generated a set of balanced functions, with up to 10 functions for each number of qubits from $n = 2$ to $n = 15$, and ran the circuits with oracles corresponding to these functions on each quantum computers we had available.

We note that since a balanced function is correctly classified whenever the measurements are not all 0, the accuracy is expected to be very high even for noisy computers, for a very noisy computer the accuracy is expected to increase as the number of qubits increases. To account for this, when reporting accuracies we take a weighted average so that the balanced and constant functions equally impact the accuracy even though there are less balanced functions in our test cases. In Figure 1, we display the average accuracy for Deutsch-Josza on the simulators and all of the available quantum computers for a variety of values of n. Due to the considerations just discussed, we see that for most computers the accuracy does not significantly worsen as n increase.

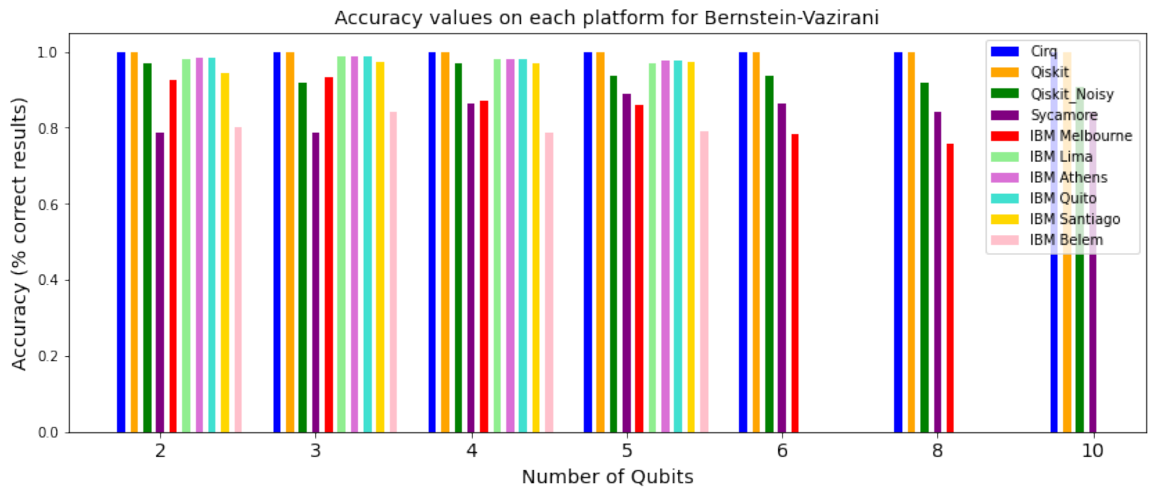

Figure 1: Here we display the accuracy for running Deutsch-Jozsa on noiseless and noisy simulators and a large variety quantum computers. For large values of n, some results are empty based on the qubit limits on the computers.

In Figure 2, we display the average run times of Deutsch-Jozsa on IBM Melbourne, IBM Lima, IBM Athens, IBM Quito, IBM Santiago, and IBM Belem. Since these computers have a limit of $n = 5$ qubits, we reports results on $n = 2$ through $n = 5$ qubits. We see that surprisingly, the run time decreases as the number of qubits increases. IBM Melbourne, the larger computer, takes the longer times, and IBM Athens and IBM Santiago are both consistently slower than the other 5-qubit computers for each value of n.

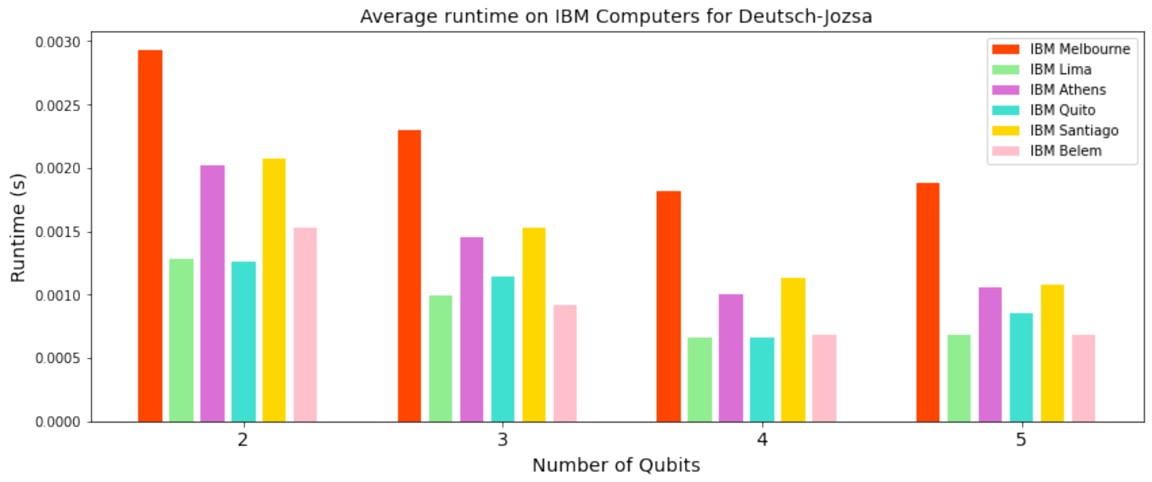

Figure 2: Here we display the average run times of Deutsch-Jozsa on a variety of IBM quantum computers for $n = 2$ through $n = 5$ qubits.

2.3 Bernstein-Vazirani

For Bernstein-Vazirani we generated a random sample of values of a and b, up to 10 example per qubit number n, and generated circuits for each function $f = a * x + b$. Like for Deutsch-Jozsa, we generated tests from $n = 2$ to $n = 15$.

In Figure 3, we display the accuracy for running Bernstein-Vazirani on noiseless and noisy simulators, and a variety quantum computers. As expected, the noiseless simulators have perfect accuracy, and the accuracies of the noisy simulator and quantum computers decrease with increasing n.

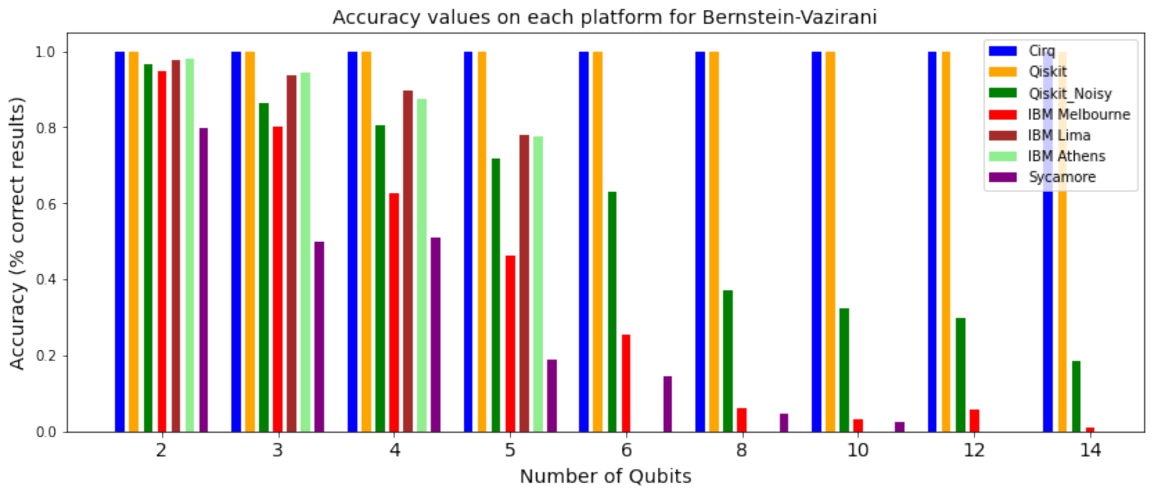

Figure 3: Here we display the accuracy for running Bernstein-Vazirani on noiseless and noisy simulators, and a variety quantum computers. For large values of n, some results are empty based on the qubit limits on the computers.

In Figure 4, we display the average run times of Bertstein-Vazirani on IBM Melbourne, IBM Lima, and IBM Athens. As in Deutsch-Jozsa, we see that for small number of qubits n, the run time decreases as the number of qubits increases. Additionally, like before, IBM Melbourne, the larger computer, takes the longer times, and IBM Athens is slower and IBM Lima. In this figure, we also see that as n gets large, the run time stops decreasing, and actually begins to just slightly increase after around $n = 6$.

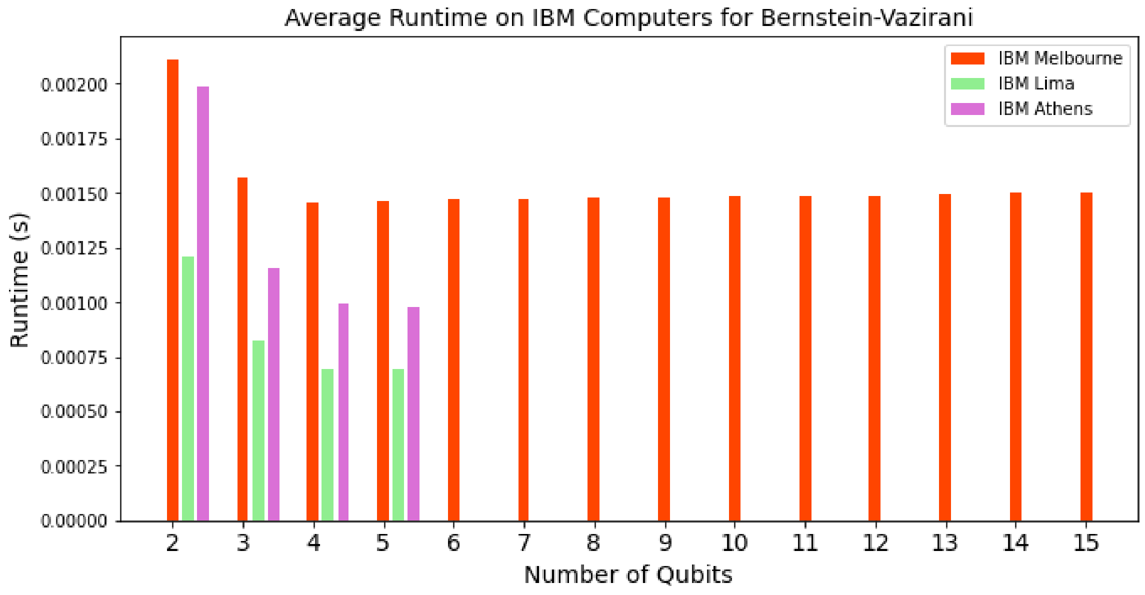

Figure 4: Here we display the average run times of Bernstein-Vazirani on a variety of IBM quantum computers for $n = 2$ through $n = 14$ qubits. Since IBM Lima and IBM Athens both have 5 qubits, we only display run times on IBM Melbourne for $n > 5$ qubits

2.4 Simon

Simon's algorithm has both as classical and quantum component, so in order to measure an 'accuracy' we repeatedly perform Simon's algorithm on equations derived from each of the computers and simulators. Simon's algorithm is designed to sample $n-1$ equations repeatedly, so given a collection of equations from the quantum computer, we randomly rearranged these results as a stream of incoming data.

In order to measure the accuracy of Simon's algorithm for a computer given our full list of repetitions, we repeatedly run Simon's algorithm on equations from the stream until no equations are left, giving us a batch of solutions. We then calculate the accuracy as the percentage of correct solutions among the batch of solutions.

We make a couple of notes about the solutions. First, due to the nature of the algorithm, Simon will always return either s or 0, as the algorithm will search for the s value, and return 0 if the guess for s is incorrect. This is because to check if a non-zero value of s is correct, one can simply check that $f(s) = f(0)$. Thus, in the case that $s = 0$ the algorithm will always find the correct s. Additionally, by using the basic heuristic that if any non-zero value is measured we return the non-zero value as our solution, this heuristic achieves near-perfect accuracy for every computer, but relies on a large number of repetitions that exceed 2^n, making the algorithm equivalent to a brute force search.

For this reason, In Figure 5, we report the percentage of solutions that find the correct s (and exclude $s = 0$) for a variety of simulators and computers.

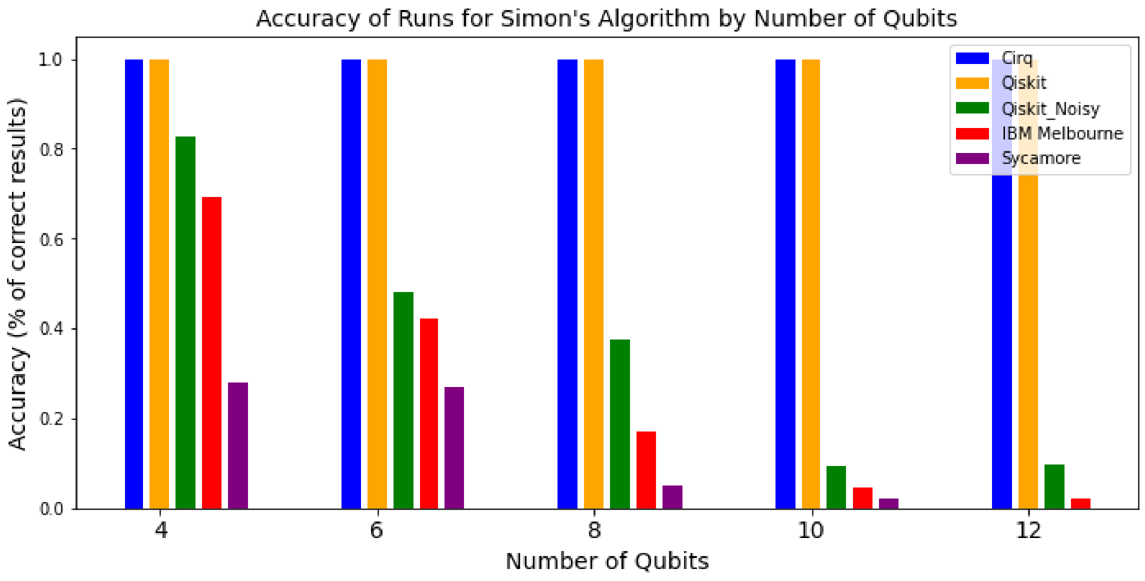

Figure 5: Here we display the accuracy for running Simon's algorithm on noiseless and noisy simulators and two quantum computers. We exclude results in the case of $s = 0$, which always gives us an accuracy of 1 due to the nature of the algorithm.

2.5 Grover

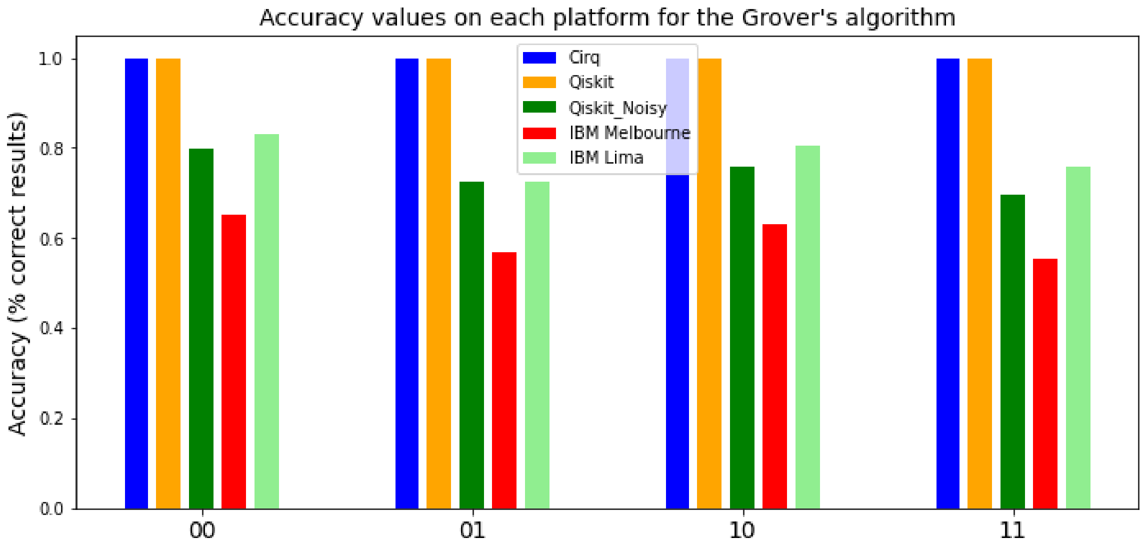

Figure 6: Here we display the accuracy for running Grover's algorithm on noiseless and noisy simulators and two IBM quantum computers. For large values of n, some results are empty based on the qubit limits on the computers.

Our ability to test Grover was limited by the design of the oracle, which, for a needle of size n, relies on a X gate controlled by n qubits. As such, for any $n > 1$ this could not be run an Sycamore, which reports an error for CCNOT gates and any other gates on 3+ qubits, and for $n > 2$ this requires a generalize control gate that is not available on any computer. So, we limit our trials for the case of $n = 2$, and report results for the four possible needles: 00, 01, 10, and 11. In Figure 6, we display the average accuracy of Grover's algorithm for the four possible needles at $n = 2$. We see that the accuracy is consistent across values of the needle.

2.6 QAOA

In the previous assignment, we implemented for QAOA for solving the MAX-CUT task, so we test QAOA on a variety of randomly generated graphs. To test QAOA, we generate five 3-regular graph at each number of even qubits n from 4 to 14, and choose 5 random values of *beta* and *gamma* for the circuit used for each graph. In order to limit error, we minimize the size of the circuit and choose $p = 1$ for all of our experiments, where p is the number of mixers and separators.

We use two metrics to measure the success of each computer: best accuracy and average accuracy. In both cases, accuracy is measured as the size of the cut divided by the size of the max cut. Since the goal of QAOA is to find the best cut on the graph, it makes sense to take many samples and choose the one that produces that largest accuracy. This metric relies on the number of repetitions taken, so for k repetitions, we repeatedly take many random samples of k repetitions, compute the best accuracy among these samples, and average over these 'best' accuracies. In Figure 8, we display the best accuracy for many simulators and computers at each number of samples taken from the full sample set in the case of $n = 10$ qubits.

We also measure average accuracy in order to get a metric for the general quality of cuts produced by the computer or simulator. In Figure 7, we display the average accuracy of the cut values produced by the QAOA max cut circuit for simulators and computers. In this figure and the previous one, we see that the quantum computers actually outperform the simulators. It is possible that this is a result of the noise, allowing the circuit to inadvertently explore more possible cut options.

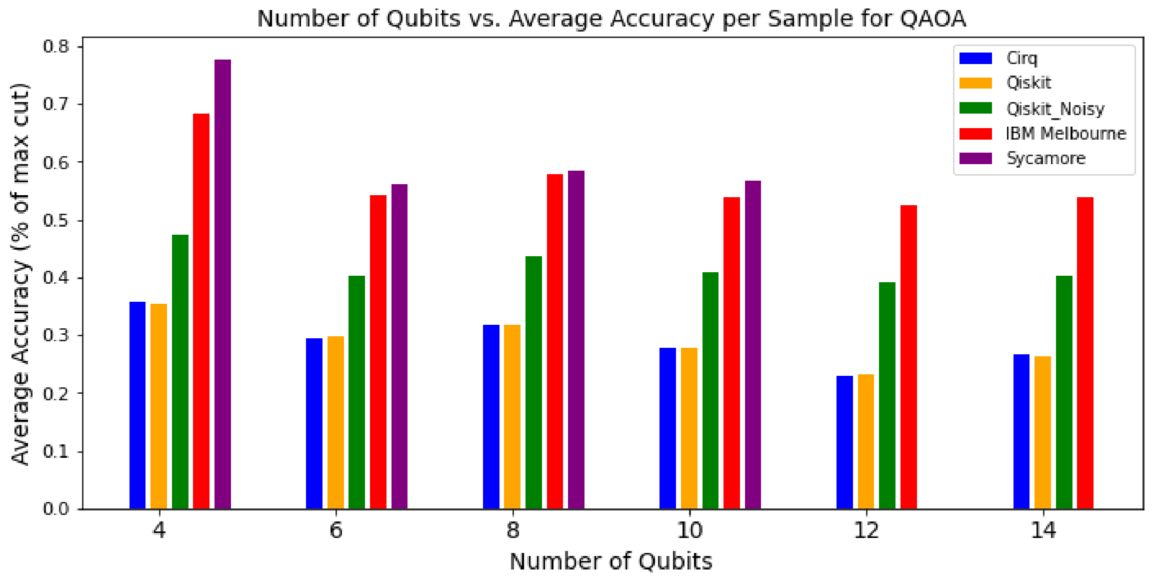

Figure 7: Here we display the average accuracy of cut values for running QAOA on noiseless and noisy simulators and two quantum computers. For large values of n, some results are empty based on the qubit limits on the computers.

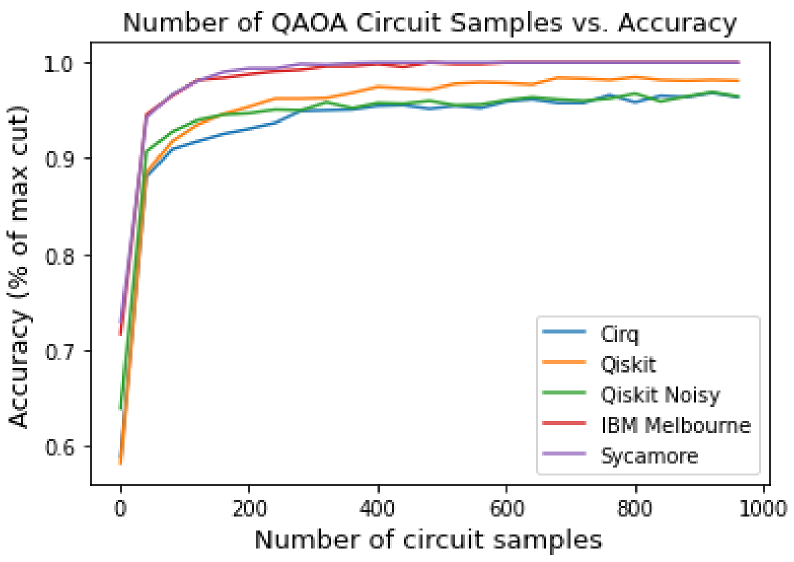

Figure 8: Here we display the best accuracy at each number of samples for running QAOA on noiseless and noisy simulators and two quantum computers, for $n = 10$ qubits. Since the QAOA algorithm relies on taking many samples of the circuit and choosing the best cut value, this figure visualizes the number of samples required to reach a high or near-perfect accuracy.

2.7 Shor's Algorithm

We previously implemented a generalized quantum order finding circuit for Shor's algorithm. It used a custom gate called ModularExp, and required 15 bits to factor the first trivial case. This is problematic for two reasons. First, we cannot run custom gates on any available quantum hardware. Second, the number of qubits we have access to is very limited. For instance, we have 12 qubits of Google Sycamore available, and one 15 qubit IBM computer. Thus, the previous implementation does not suffice.

We consider a special case of quantum order finding for N=51 and N=85. As described in [1], these numbers have a simplifying property that causes their orders to be a power of two. Because of this, ModularExp can be greatly simplified: each number can be factored with only 8 qubits using CNOTs instead of ModularExp. We based our implementation on the one described in [1].

There exists four possible ModularExp implementations for different values of a depending on the value of N. For each of these four circuits, quantum order finding leads to is different, unique order r. In our experiments, for each N we select two values of a for each circuit type. In Table 1 we display the associations between circuit type and each value a used in our experiments.

N	Circuit A	Circuit B	Circuit C	Circuit D
51	16, 35	4, 47	2, 49	5, 29
85	16, 69	4, 81	36, 59	22, 73

Table 1: Here we display the associations between the values of a used in our experiments and circuit type, where each circuit has a different implementation of ModularExp.

For each pair (N, a), we run the circuit over many repetitions and measure the percentage of repetitions that result in the correct order r, and report our results for $N = 51$ and $N = 81$ in Figures 9 and 10, respectively. We note that as we would expect, values of a belonging to the same circuit type give similar average accuracies. We planned to run the Shor circuit on Google Sycamore, but the system could not handle the inverse Quantum Fourier Transform (QFT) gate.

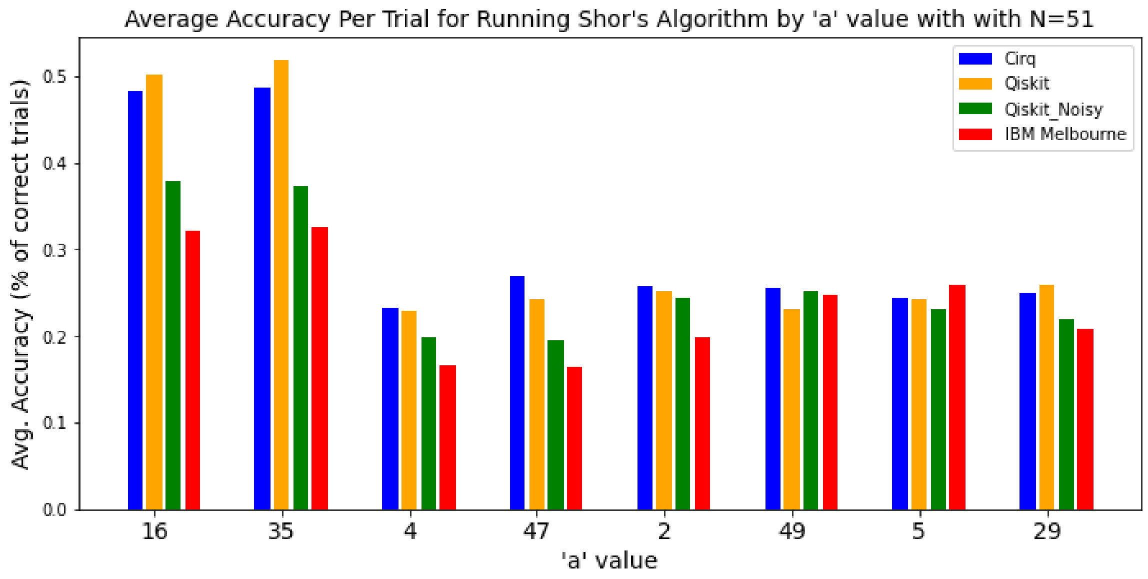

Figure 9: Here we display the average accuracy of Shor's Algorithm on simulators and IBM Melbourne for N=51. We note that here, if no order of found we consider the trial a failure; if only incorrect orders were considered as failures, our accuracy values would be significantly larger.

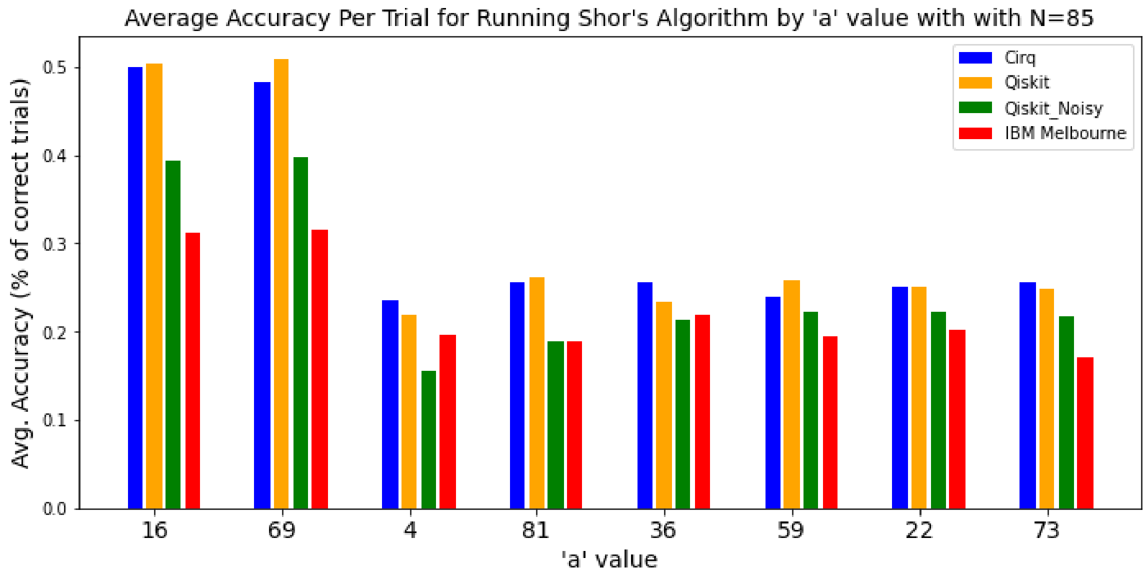

Figure 10: Here we display the average accuracy of Shor's Algorithm on simulators and IBM Melbourne for N=85.

2.8 Analysis and Comparison of Quantum Computers

Here we provide a variety of analysis for the results on the quantum computers.

Scalability in n, Accuracy For most of the algorithms in which we measure accuracy for varied qubit numbers n, as expected, the accuracy decreases as the number of qubits increases. This makes sense because as we increase the size of the circuit, there is a greater chance if error being introduced. One exception is Deutsch-Jozsa, in which the accuracy of classifying balanced functions becomes easier as n inceases, as the change of accidentally outputing all 0s decreases.

Scalability in n, Time Surprisingly, we see that as the number of qubits increases, the run time of the circuits actually decreases. In Figure 2 We were only able to analyze run time over small numbers of qubits n due to the limited size of the computers, so we expect that if n were to get large, the run time would then begin to increase as n increases. We see hints of this in Figure 4, in which the run time stops decreasing for IBM Melbourne for large n, and eventually begins to just slightly increase.

Comparison of Computers and Simulators, Accuracy With the exception of QAOA, we see that Sycamore and IBM Melbourne, the larger computers, tend to have more error, and Sycamore especially has a very low accuracy. This makes sense because this computers have a significant amount of noise, making a correct result less likely, especially in algorithms such as Bernstein-Vazirani and Grover in which exactly one of the possible outputs is classified as correct. These results suggest that for large circuits with run on many qubits, error corrections is very important in order to be able to get even a small percentage of accurate results.

From the experiments it is clear that the noiseless simulators attain significantly better results than the quantum computers, which is to be expected because they do not suffer from any noise, and confirms that the low accuracies on the quantum computers at high numbers of qubits are a result of noise in the computers and not issues with the implementation. We also note that the Qiskit noisy simulator provided an accurate and useful approximation of the IBM quantum computers, with results that were very closely within the range of those attained by the computers.

Comparison of Computers, Time We were able to measure execution time for many of the IBM computers. In general, all of the computers run the experiments very quickly, which makes sense because all of our circuits are relatively small. We see consistently that IBM Melbourne, the computer with the most qubits, is the slowest. There is some additional variation within the other computers, but overall there run times between the computers are fairly consistent.

3 Error Correction

We implemented bit flip error correction in Cirq, included in the file `error_correction.py`. Our implemented and experimentation with error correction was limited due to practical considerations described in Section 3.1. In Figure 11, we display a visualization of the encoding and correction stages of the error correction circuit for a single qubit, produced by Qiskit. Here the encoding and correction are separated by a Y gate, which can represent any one-qubit operation.

As explained in Section 3.1, we only do error correction at the end of the circuit, and since the phase of a quantum state does not affect the measurements, performing phase flip error correction would not impact our results. As such, in this report we focus on bit flip error correction. We also note that as we do error correction at the end of the circuit, we are able to measure all qubits and perform the decoding step classically following the bit flip error correction algorithm. For error correction experiments, we use the same tests as those described in Section 2.1.

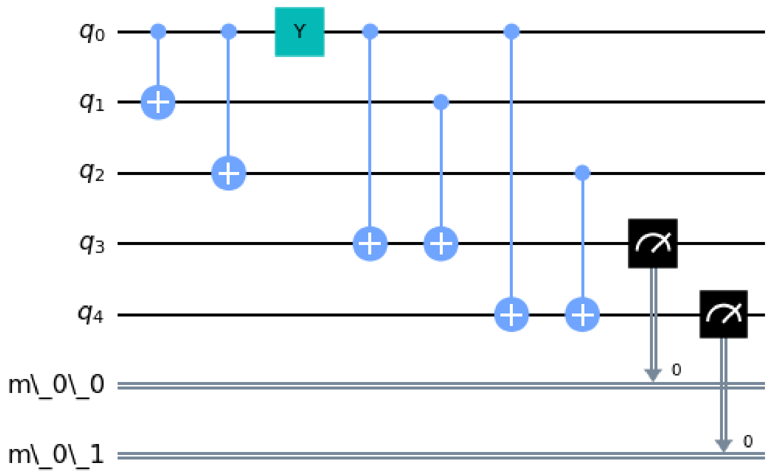

Figure 11: A visualization of the encoding and correction stages of the error correction circuit for a single qubit. Here the encoding and correction are separated by a Y gate, which can represent any one-qubit operation.

3.1 Practical Considerations

Here we describe some practical considerations that limited our error correction efforts.

Mid-Execution Error Correction In our approach, we only perform error correction at the end of the circuit. This is because if we were to perform error correction in the middle of the circuit, we would have to dynamically add X gates based on the measurements of the extra helper qubits and reset these extra helper qubits to 0. Because any gates must be invertible, it wouldn't be possible to always reset the helper bits to 0.

Shor Code Shor code corrects for both bit flip and phase flip errors at the same time, but in doing so it requires a 9 fold increase in the number of qubits, making experimentation with Shor's code impractical given our qubit limit constraint. Speficially, the smallest quantum circuit we ran required 2 qubits, and error correction with Shor's code would increase this simple circuit to 18 qubits. This exceeds the number of qubits on any hardware available to us. Additionally, 18 qubits is too many to simulate on the classical CPUs we had available. We therefore corrected bit flip error correction.

3.2 Error Correction Results

We ran error correction for the Bernstein-Vazirani and Grover circuits. We note that our runs were limited because of the qubit limitations; since IBM Melbourne had a 15 qubit limit, we were only able to run error corrected circuits that used at most 3 qubits. In Figure 12 and 13 we display the average accuracies produced by the original and corrected circuits, where the accuracy is the percentage of repetitions of the circuit that gave the correct output. In both cases, we see that the error corrected circuits did not improve the accuracy of the circuit, and we explore this issue in the following section.

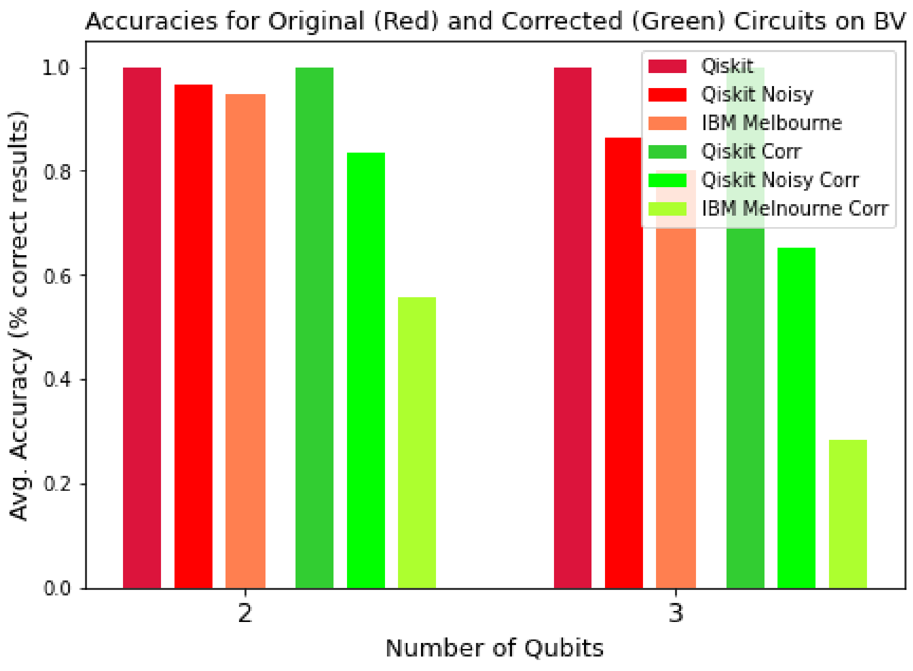

Figure 12: Average Accuracy of the Bernstein-Vazirani original and corrected circuit on a non-noisy Qiskit Simulator, noisy Qiskit simulator, and IBM-Melbourne for n=2 and n=3 qubits. Here the accuracy is the percent of repetitions of the circuit that recovered the correct value a for $f(x) = a * x + b$.

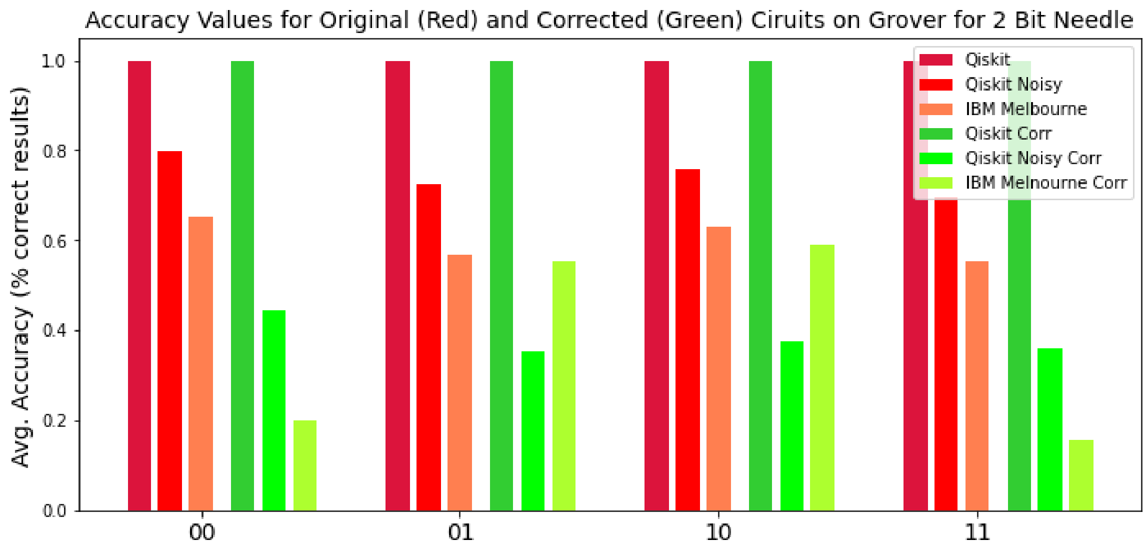

Figure 13: Accuracy of the Grover original and corrected circuit on a non-noisy Qiskit Simulator, noisy Qiskit simulator, and IBM-Melbourne for n=3 qubits. Here the percent of repetitions of the circuit that recovered the correct needle.

3.3 Error Correction Analysis

As we see in Figure 12 and 13, adding bit flip error correction to our circuits did not make the accuracy better, and in many cases made it much worse. This has two likely causes. First, in order to fit within the gate sets in the quantum computers, these computers likely had to significantly increase the number of gates used by the error corrected circuits, making them highly prone to error. Second, while we performed bit flip correction on the circuits, the increase in the number of qubits in the error correction circuits made phase flip errors far more likely, and these phase flip error could have significantly worsened the accuracy of these circuits.

4 Experience

When creating the oracles used by many of the circuits in the previous assignments, we analyzed the algorithms using linear algebra and identified the proper construction of the gates as matrices, so to construct the oracles we created custom matrix gates. It is difficult for quantum computers to properly map general unitary matrices, so these computers do not accept custom matrix gate, and so we had to re-construct our oracles using common gates accepted by the computers. For a couple of these changes, we directly added implementations of Oracles from Cirq tutorials and cited the tutorials in our notebooks. The changes we made include:

DJ/BV/Simon Re-designing the U_f oracle using common gates.

Grover Re-designing the Z_0 and Z_f oracles using common gates.

Shor Re-designing `ModularExp` class using common gates.

We also had to change the structure of our python notebooks. Because the quantum computers take a long period of time to execute, having the results and experiments in the same notebook was infeasible, as a single run through the notebook would take over a day to give results. So, we separated the executions and results elements so that after executing and saving the Job ID's, we could still use and restart the results notebook without losing any information.

For Simon's algorithm, we had to change the implementation of the running of the algorithm in order to be able to accept a large collection of quantum computer results at one. Simon's algorithm is designed to sample $n-1$ equations repeatedly, so given a collection of equations from the quantum computer, we randomly rearranged these results as a stream of incoming data. More details for this are provided in Section 2.4 in which we describe the testing and results for Simon's algorithm.

In general, we found IBM's API to be significantly easier to navigate, and made it very easy to read in results. On the other hand, Sycamore was hard to navigate, and we had to rely heavily on the sample code provided by the instructors. This is likely a result of IBM having a far more public facing API than Sycamore.

References

[1] Michael R Geller and Zhongyuan Zhou. Factoring 51 and 85 with 8 qubits. *Scientific reports*, 3(1):1–5, 2013.