Solving a 3x3 Sudoku using Grover's Algorithm implemented with Qiskit and IBM Q

Noah Hamilton, U95110302
University of South Florida, hamiltonn@usf.edu

***Abstract* – Quantum algorithms are the key to furthering technology in the realm of computing. They can greatly reduce the complexity of search problems and other common computer science challenges. In this paper, Grover's Algorithm which is designed to reduce the time complexity of decision problems is explored. It is illustrated using the 3x3 Sudoku Binary problem and implement using Qiskit with the IBM Q Simulator.**

Key Terms – Quantum Computing, Grover's Algorithm, Qiskit, Search Problems

Introduction and Motivation

Grover's algorithm is a quantum algorithm that finds unique solutions that produce unique outputs for search problems. Grover's algorithm used use O ($\sqrt{N}$) evaluations instead of the typical O(N) evaluations from a classical computer. This is super beneficial as it can drastically speed up performance for larger domain sizes. Like other quantum algorithms, Grover's algorithm gives the correct answer as probability less than one and with more iterations can increase the probability of the correct output. The goal is to demonstrate this quantum supremacy on an unstructured search.

Problem

This demonstrates this improved speed time using a Sudoku problem with a 3x3 matrix. There is a 2x2 example using 4 qubits and implementing Grover's Algorithm, but not the 3x3 version. The problem of the 3x3 Sudoku Binary Sudoku is defined as such and can be seen in Figure 1.

```
# Problem Environment + Rules
# Each Column must contain exactly one 1.
# Each Row must contain exactly one 1.
# ----------------
# | v0 | v1 | v2 |
# ----------------
# | v3 | v4 | v5 |
# ----------------
# | v6 | v7 | v8 |
# ----------------
```

Figure 1: 3x3 Binary Sudoku layout and rules for the problem.

Each of the 9 squares will hold a binary value and the goal of the game will be to find the solutions where every column and every row must have exactly one 1 appear. The objective of this term project is to create an oracle to solve this problem using Qiskit and IBM Q Simulator using Grover's Algorithm.

Expected Path to Completion

During this term project, I will start by expanding on the process of Grover's Algorithm by defining my number of qubits and the problem conditions. For this case, I have 9 binary squares, so that leads to 2^9 possibilities or 512. That means if I were to brute force this, I would have to check 512 cases until I found the right solution. Since I am using the Grover's algorithm, I only need to iterate the square root of 512 so 22 times to find the correct solution. After defining my problem, I will create 9 qubits and apply a Hadamard Gate to create a Superposition to prep for applying the algorithm. I will then create the conditions that I need to check to satisfy a correct solution to the game. Once I identify the solution, which there should be 6 possibilities for a 3x3 game out of 512 possible combinations of ones and zeros. I will then create an oracle with the solution states by introducing a negative phase to the marked states. Once I do this, I will run several iterations to solve the Oracle using Grover's Algorithm and then evaluate my results with plots and simulations that Qiskit provides. I hope to explore the eccentricities of this algorithm through this complicated search problem.

Background

To be able to understand the process behind implementing this algorithm, it is important to grasp concepts in Quantum Computing. These basic concepts include Qubits, Quantum Algorithms, Quantum Circuits, and Quantum Logic Gates. After seeing these basic concepts, the implementation of Grover's Algorithm can easily be understood.
Quantum information is stored in qubits as opposed to classical bits.[5] Classical bits are known as either 0 or 1 and are strictly these values. Qubits on the other hand, are of the form: $|\varphi\rangle = \alpha|0\rangle + \beta|1\rangle$, where $|\alpha|^2 + |\beta|^2 = 1$. Alpha and Beta are the square roots of the probability of the qubit collapsing to 0 and 1, respectively. Measurement operator on a qubit has two possible outcomes, which are associated with the value of a classical bit (Figure 2).

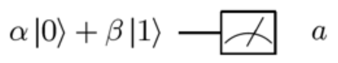


Figure 2: Measurement collapsing qubit to classical 0 or 1.

Quantum algorithms are basically a “set of instructions for a quantum computer”. Two types of operations are allowed on qubits within a quantum algorithm: measurement and quantum state transformation. All the operations must be unitary and reversible to not lose information within the qubits. Most quantum algorithms act like a ‘black box’ that takes inputs and produces outputs with the goal being to reduce the number of queries on each input. Grover’s Algorithm is an example of a Quantum algorithm that drastically reduces the number of queries on inputs over a classical system. This implementation of Grover’s Algorithm is on an unstructured search problem.

An unstructured search problem is defined as a problem where nothing is known, or no assumption is used about the structure of the solution space and problem statement.[2] A classic example of an unstructured search problem is as follows: Consider the problem of searching for a phone number in an unsorted directory with N names. To find someone’s phone number with a probability of ½, any classical algorithm will need to look at least N/2 names. Basically, a classical search looks at each name and tests if it is true or false. The general unstructured problem with a search space N required O(N) evaluation classically; however, with a Quantum Computer and Grover showed that is can be solved with the bounded probability of O(sqrt(N)). [2]

Before diving into the specifics of Grover’s Algorithm, it is important to see how a Quantum Circuit works and Quantum Gates. In general, circuit equivalences help to “analyze complex processing blocks or to explain different logical operations. Conventions from classical logic circuits are taken into the quantum circuits such as the wires carrying signals and carrying quantum states to different parts of the circuit. Figure 3 below shows an example of classical circuit (a half adder) and a generic Quantum Circuit.

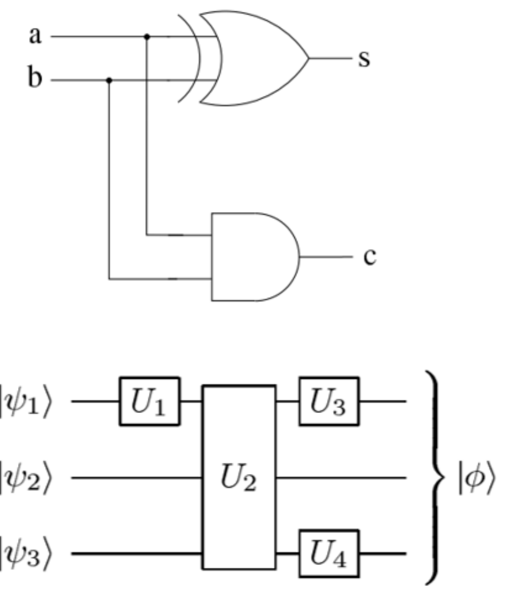
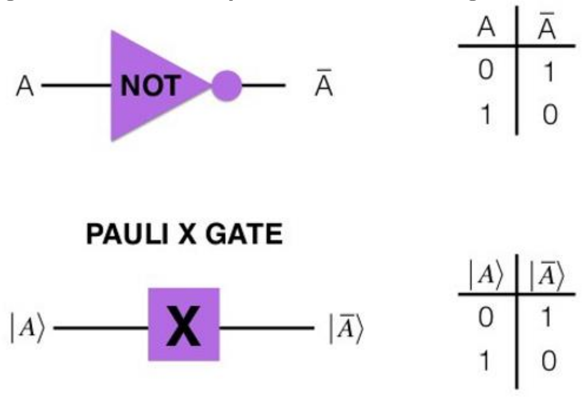


Figure 3: Classical Circuit (half adder) (top) and Generic Quantum Circuit (bottom)

As mentioned previously, Quantum algorithms rely on quantum state transformations, which are essentially matrices that are applied to a quantum bit. The classical gates AND, OR XOR, and NOT are not all implemented within quantum gates as not all these operations are reversible, which is required for quantum algorithms to run. The workaround for this reversibility clause is that we used extra bits, often referred to as ancilla bits that are corrupted, but act as an output for the classical gates. Because the solution to the binary problem relies on these classical gates and their quantum equivalents, below is a summary of each. Each of these gates can be represented by a matrix, which is included below.

The classical NOT gate can be represented by the Quantum X gate, which has a very similar affect, see Figure 4 below.



A	$\bar{A}$
0	1
1	0

$\lvert A\rangle$	$\lvert\bar{A}\rangle$
0	1
1	0

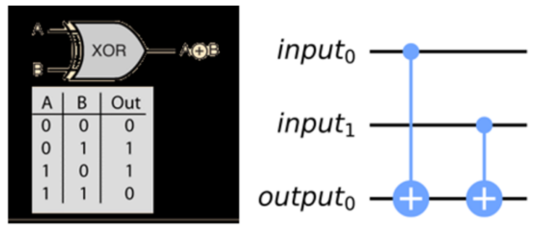
Figure 4: Classical NOT gate and truth table (top) compared to Pauli X Gate in quantum (bottom).

The Pauli X Matrix shown in Figure 5 operates on just one qubit like the NOT gate in classical logic.

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

Figure 5: Pauli X Matrix

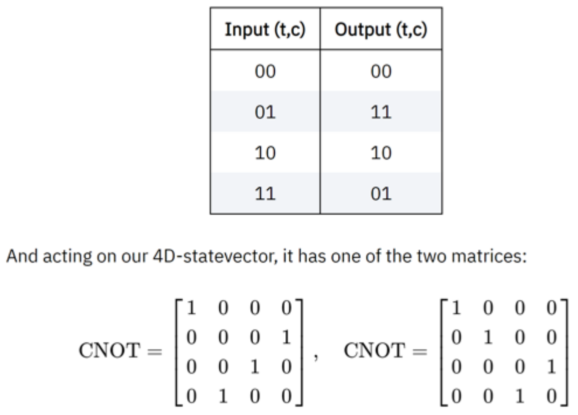
The next gate that is important in the implementation of Grover’s Algorithm for this case is the classical XOR gate or the CNOT gate in Quantum. For this XOR case in the solution to our problem, one ancilla bit was used for the output and this has the effect of not impacting input bits.



A	B	Out
0	0	0
0	1	1
1	0	1
1	1	0

Figure 6: Classical XOR Operator (left) and 2 CNOT gates to construct a quantum XOR (right).

The CNOT is known as the control not where there is an operating bit and a control bit. If the control bit is 1, then the operating bit is flipped, else the operating bit remains the same. This can be seen in Figure 7 below.



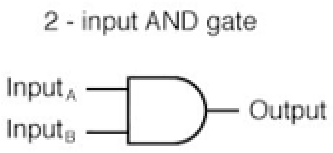
Input (t,c)	Output (t,c)
00	00
01	11
10	10
11	01

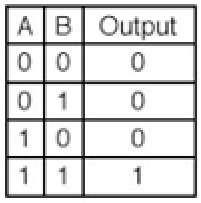
And acting on our 4D-statevector, it has one of the two matrices:

$$\text{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}, \quad \text{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

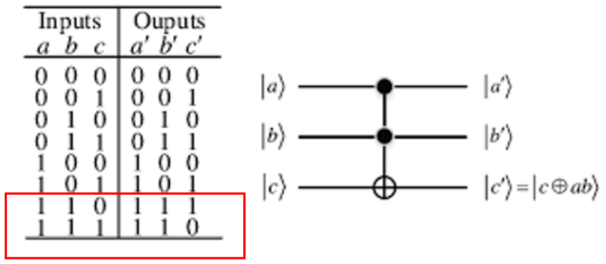
Figure 7: Truth table for CNOT gate and respective matrices.

Next, we have the classical AND gate, which will be represented by the Toffoli gate or the CCNOT gate.
Like how the CNOT gate works, the CCNOT gate uses the first two bits as a control and the third bit as the operating bit. This has the same affect as the classical AND gate with an ancilla bit added as the third bit for output. Figure 8 shows a classical AND gate compared to a Toffoli Gate.





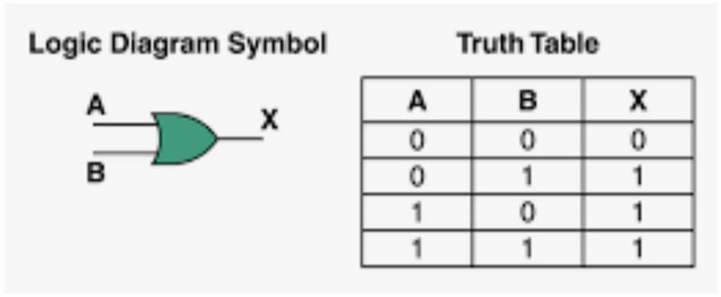
A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1



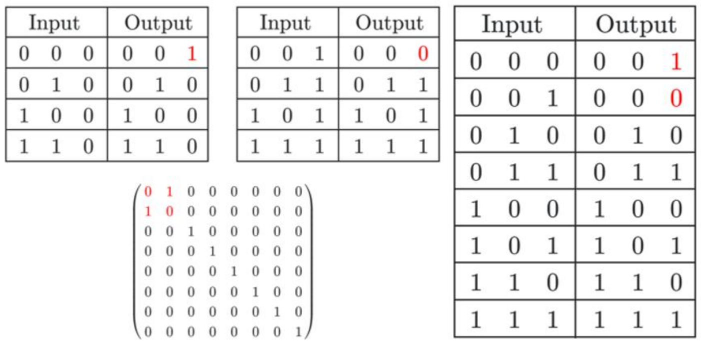
Inputs			Ouputs		
a	b	c	a'	b'	c'
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	1	0	1
1	1	0	1	1	1
1	1	1	1	1	0

Figure 8: Classical AND Gate (top) and Quantum Toffoli Gate and truth table (bottom)

The last classical gate that is required for the solution to this specific sudoku binary problem in the 3x3 case is the classical OR gate. There is little to none current literature and research about the classical OR gate represented as a traditional quantum gate. However, with some help from Quantum Stack Exchange [9], a unitary matrix can be defined with ancilla bits to simulate the effect of an OR gate. Figure 9 shows the classic OR operation along side of the computed matrix to simulate this same operation on a quantum system. Note: This or operation is only shown as a unitary matrix in the circuit and does not use pre-defined gates.



A	B	X
0	0	0
0	1	1
1	0	1
1	1	1



Input			Output		
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	0	1	1	0

Input			Output		
0	0	1	0	0	0
0	1	1	0	1	1
1	0	1	1	0	1
1	1	1	1	1	1

Input			Output		
0	0	0	0	0	1
0	0	1	0	0	0
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	1	0	1
1	1	0	1	1	0
1	1	1	1	1	1

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

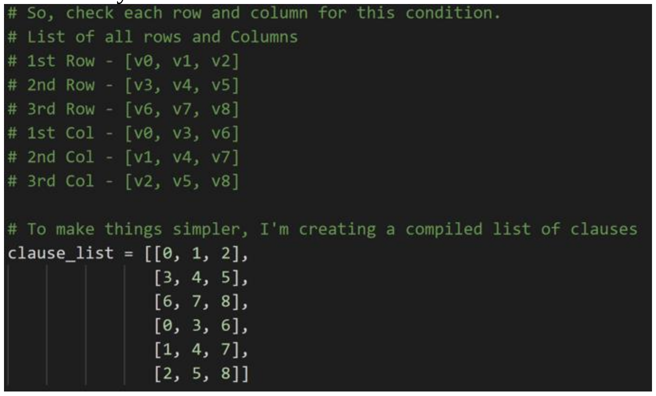
Figure 9: Classical OR operation (top) and custom Quantum Operation with Unitary Matrix with ancilla bit added.

Now that all the classical gates that are needed have a Quantum Equivalent, there is enough information to proceed to how the Grover's Algorithm works.
Grover's Algorithm requires the problem to be turned into a quantum circuit and then the solutions to be made into an oracle function. After this oracle is used to identify the correct solution, it then flips the phase by 180 degrees. After the phase flip, there is an amplitude amplification about the mean. The process is applied over square root of N times where N is the power of two of number of qubits. For example, this binary sudoku problem has 9 qubits, so 2^9 = 512 and sqrt (512) is approximately 22, so this oracle, reflection, and amplification needs to occur 22 times to produce an accurate result. This result produced should identify solutions to the problem based on the oracle. In this case, there are exactly 6 solutions of out 512. In the 2 by 2 case, there are exactly 2 solutions out of 16. Since the basics of Grover's algorithm have been explored, the implementation can be clearly understood.

SOLUTION/IMPLEMENTATION

To be able to understand the process behind implementing this algorithm, it is important to grasp concepts in Quantum. The first step is to identify the conditions that are required for a winning state.

From observation, six restrictions are in place for this binary sudoku. These conditions are shown below in Figure 10 in the code. Essentially, it is just each row, and each column must have exactly one 1. It is easiest to pass this to the solver via an array shown below as well.



```python
# So, check each row and column for this condition.
# List of all rows and Columns
# 1st Row - [v0, v1, v2]
# 2nd Row - [v3, v4, v5]
# 3rd Row - [v6, v7, v8]
# 1st Col - [v0, v3, v6]
# 2nd Col - [v1, v4, v7]
# 3rd Col - [v2, v5, v8]

# To make things simpler, I'm creating a compiled list of clauses
clause_list = [[0, 1, 2],
               [3, 4, 5],
               [6, 7, 8],
               [0, 3, 6],
               [1, 4, 7],
               [2, 5, 8]]
```

Figure 10: Python code for conditions of each column and row to be checked.

A truth table was created with the check for each row or column. This can be seen in Figure 11 and only represents one check through. Since finding exactly one 1 is not an easy task, a custom circuit implementation had to be created.

A	B	C	Out
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

Figure 11: Truth Table for checking one specific row or column for a correct solution.

The equivalent classical circuit was derived from this truth table as shown in Figure 12 below.

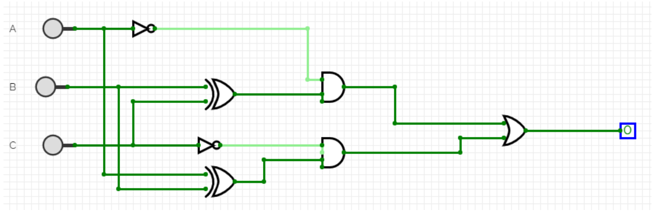


Figure 12: Equivalent Circuit based on the Truth Table created in Figure 11 for the Binary Sudoku 3x3 solution.

It is important to note that this truth table and equivalent circuit just check one row or column and that each of the rows or columns defined in the "clause_list" in Figure 10. For one row, the classical logic boils down to the logical statement in Figure 13.

((NOT A) AND (B XOR C) OR ((NOT C) AND (A XOR B))

Figure 13: Classical Logic for solution to exactly 1 input being true.

Using the Quantum Gates defined above, a quantum equivalent circuit was constructed to represent the solution to one of these rows to be used as the oracle in Grover's Algorithm. Figure 14 shows the solution finder for one row as a Quantum Circuit. Three qubits were used for input, 4 extra bits used for operations, and one output qubit for the correct answer. This is the first model of the circuit in Quantum Circuits. It is possible to reduce this complexity, but time did not allow for that for this implementation.

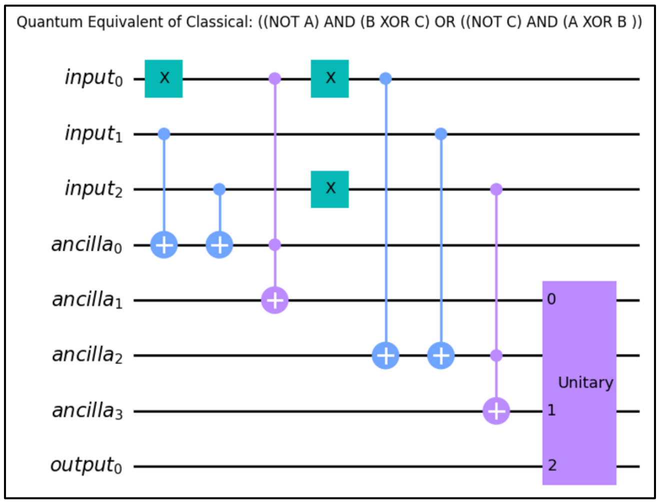


Figure 14: Quantum Equivalent Circuit for the classical circuit shown in Figure 12 and logic shown in Figure 13. *Note: The Unitary purple block is the OR operation matrix.* Image drawn by MATPLOTLIB in Python Code.

After creation of the Quantum Equivalent Circuit function, then it was time to implement this as a full oracle for the entire set of 9 qubit inputs and six clauses. To complete this task an extra 24 ancilla qubits were needed and this makes the computation quite heavy. This is not ideal, and the complexity could be reduced by reusing an ancilla bit by performing the same operation again after one output is determined and not having to create another ancilla each time. This solution does not implement that as it was outside of the scope of what is being attempted to show.

For the creation of the full oracle, each of the clauses in the clause list are passed into the Quantum Equivalent Circuit function and this allows the function to create a larger circle that will be known as the oracle. Before those 9 qubits are fed into the function, the function also runs twice to reset the value back to the zero state for measurement at the end of the circuit. Figure 15 was generated to show the complete oracle circuit.

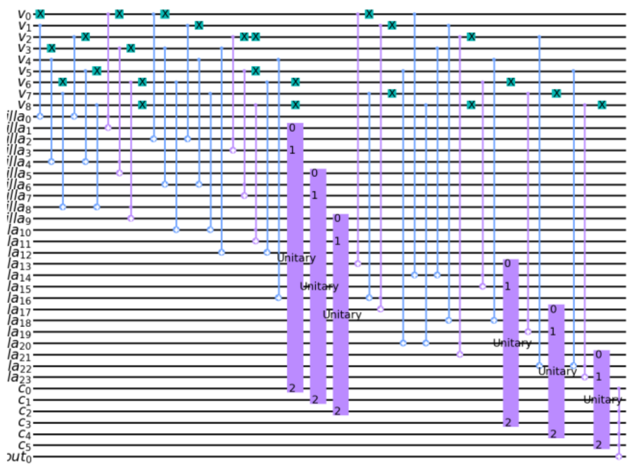


Figure 15: Generated Quantum Circuit from Qiskit library for the oracle.

After the oracle is created, the Quantum Algorithm is almost complete. The remaining steps are to initialize the input qubits into the super position state which is done with Hadamard gate. The Hadamard gate is represented by the matrix shown in Figure 16 and the quantum single qubit circuit equivalent of Figure 17.

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

Figure 16: Hadamard gate that is applied to each qubit before any quantum state transformations are performed.

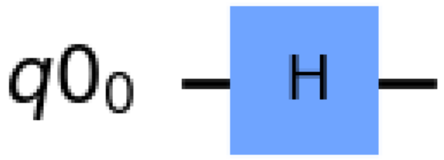


Figure 17: Hadamard Gate representation on a single qubit.

As mentioned in the Background section, the Oracle is applied repeatedly about 22 times to perform the amplitude amplification of the marked states.

RESULTS/CONCLUSIONS

Using Python3, the Qiskit Library, and the IBMQ Simulator, the full algorithm was run. Initially, there were several errors regarding syntax and implementation. After fixing initial bugs, there was a Qiskit Implementation Error thrown, this was the max qubits reached error. 40 qubits were the initial number of qubits used on the simulator. After further research, IBMQ open source QASM simulator only supports up to 32 qubits. Knowing this, the function that solved the puzzle needed to be reduced. This function was reduced by 3 ancilla bits per clause, which brought the total number of qubits down from 40 to only 28 qubits. The full circuit was drawn and running; however, the circuit diagram was too large to include or even read. The IBMQ simulator timed out each time after about 3 hours running. The error given was with regards to reducing complexity, which is a future goal of this algorithm.
Based on the research regarding the implementation with 4 qubits and a 2 by 2 sudoku the output plot should have resembled the Figure 18 below, but with some changes.

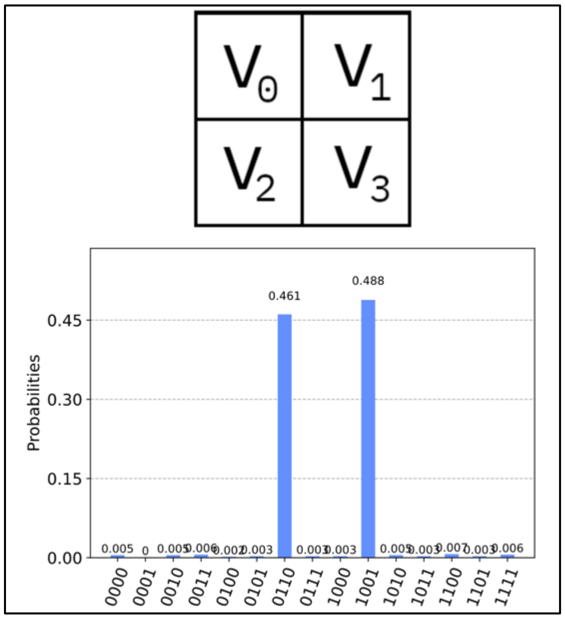
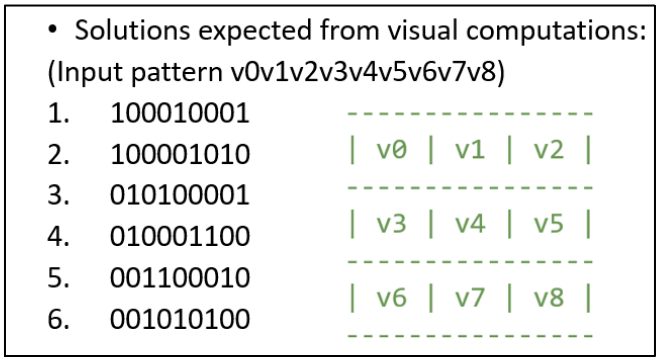


Figure 18: Previous work and solution on a 2 by 2 system

First, there would 512 possible combinations of 9 qubits measured. Second, there are a total of 6 solutions, so the amplitude of each of these solution states would be approximately 3/256 = 0.0117. The solution states are shown below in Figure 19.


- Solutions expected from visual computations:

(Input pattern v0v1v2v3v4v5v6v7v8)

1. 100010001
2. 100001010
3. 010100001
4. 010001100
5. 001100010
6. 001010100

```
----------------
| v0 | v1 | v2 |
----------------
| v3 | v4 | v5 |
----------------
| v6 | v7 | v8 |
----------------
```

Figure 19: All the 6 possible solutions for the 3 by 3 Binary Sudoku problem posed as binary strings.

Despite being unable to fully simulate this circuit using Grover's Algorithm, it can still be seen that this is a practical algorithm for solving many unstructured search problems. Current literature [10] and [11] show solutions to the Polynomial Root Finding Problem and Transcendental Logarithm Problem. Overall, Grover's search algorithm is quite useful, and many more applications are on the horizon.

REFERENCES

[1] https://qiskit.org/textbook/ch-algorithms/grover.html

[2] Borbely, E., "Grover search algorithm", *arXiv e-prints*, 2007.

[3] Chapter 6 Quantum Algorithms – Professor Chen USF

[4] https://en.wikipedia.org/wiki/Grover%27s_algorithm

[5] Garcia-Escartin J.C., Charmorro-Posada P. "Equivalent Quantum Circuits", *arXiv:1110.2998v1* [quant-ph] 13 Oct 2011.

[6] IBM Quantum. https://quantum-computing.ibm.com/
, 2021

[7]https://qiskit.org/textbook/ch-states/atoms-computation.html

[8]https://towardsdatascience.com/demystifying-quantum-gates-one-qubit-at-a-time-54404ed80640

[9] https://quantumcomputing.stackexchange.com/a/5834

[10] Y. Tang and S. Su, "Application of Grover's Quantum Search Algorithm to Solve the Transcendental Logarithm Problem," 2014 Tenth International Conference on Computational Intelligence and Security, 2014, pp. 445-449, doi: 10.1109/CIS.2014.166.

[11] G. Sun, S. Su and M. Xu, "Quantum Algorithm for Polynomial Root Finding Problem," 2014 Tenth International Conference on Computational Intelligence and Security, 2014, pp. 469-473, doi: 10.1109/CIS.2014.40.