Implementation of Grover’s Algorithm on 3x3 Puzzle

Quantum Computing and Communication Term Project

Noah Hamilton

Spring 2021

Outline

- Qubits/ Quantum Algorithm
- Background on Grover's Algorithm
- Problem
- Implementation
- Further Application

Qubits

- Quantum Information is stored in Qubits. The classical counterparts are known as classical bits and can be 0 or 1.
- Qubits are defined as a superposition of the states in the form:
- Measurement is used to get the associated classical value with alpha and Beta Probability.

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle, \tag{1}$$

where $|\alpha|^2$ and $|\beta|^2$, such that $|\alpha|^2 + |\beta|^2 = 1$, are the respective probabilities of finding $|0\rangle$ and $|1\rangle$ after a measurement in the $\{|0\rangle, |1\rangle\}$ basis. Figure 2 shows the circuit representation of a measurement. A measurement on a qubit has two possible outcomes which can be associated to the binary value of a classical bit.

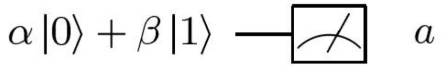
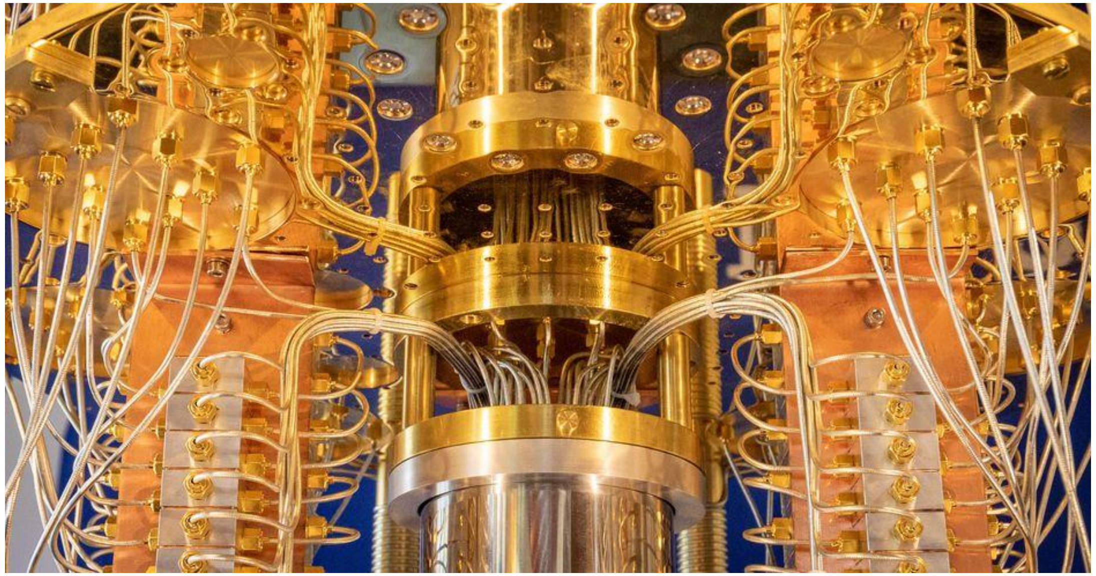


FIG. 2: Measurement in a quantum circuit. The binary outcome of the measurement can be associated to a classical bit a, which takes value 0 if the state $|0\rangle$ is found (with probability $|\alpha|^2$) and takes value 1 when the state is $|1\rangle$ (with probability $|\beta|^2$).

https://arxiv.org/pdf/1110.2998.pdf

Quantum Algorithm

- “Set of Instructions for a quantum computer”
- 2 Types of Operations allowed: measurement and quantum state transformation, operations themselves must be unitary (reversible)
- The goal is to create a “black box” known as an oracle that takes inputs and outputs a state with the minimum number of queries.



https://arxiv.org/ftp/arxiv/papers/0705/0705.4171.pdf

https://www.cnet.com/news/ibm-new-53-qubit-quantum-computer-is-its-biggest-yet/

Unstructured search

- A problem where nothing is known or no assumption is used about the structure of the solution space and the statement.
- Classic Example: Consider the problem of searching for a phone number in an unsorted directory with N names.
- In order to find someone's phone number with a probability of ½, any classical algorithm will need to look at least N/2 names.
- Simple terms, classical search looks at each name and tests true or false.

For a search space of size N, the general unstructured search problem requires O(N) evaluation of f.

On a quantum computer, however, Grover showed that the unstructured search problem can be solved with bounded probability within $O(\sqrt{N})$ evaluation of f.

https://arxiv.org/ftp/arxiv/papers/0705/0705.4171.pdf

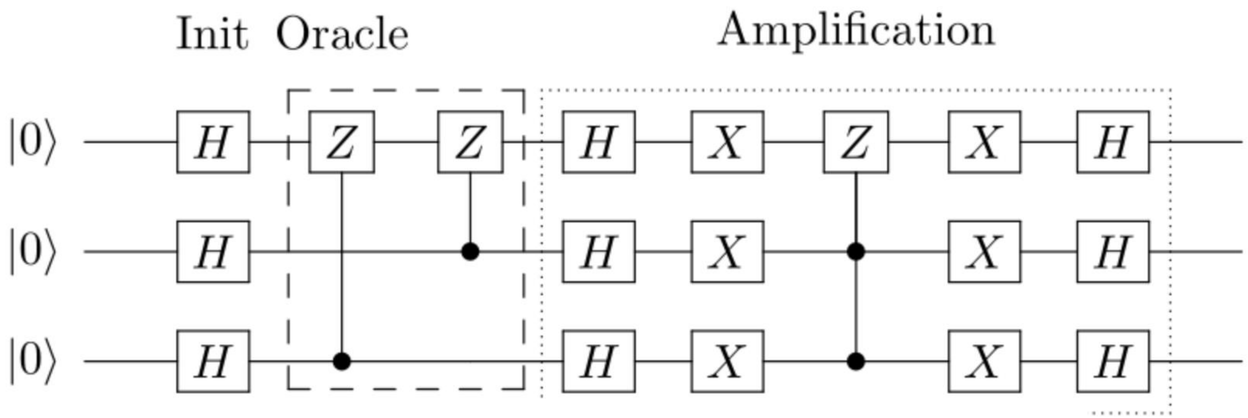
Background on Grover’s Algorithm

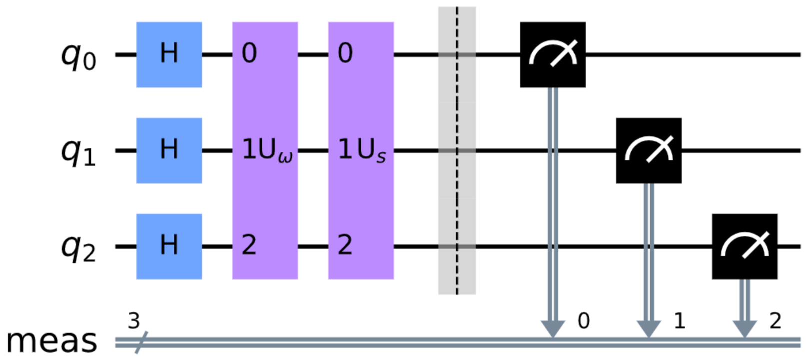
- Used to solve unstructured search problems
- Phase Negation on Marked States, then inversion about the mean.
- Uses Amplitude Amplification to speed up search problems Quadratically
- Apply an Oracle (which identify the qualifying states) and Diffuser over sqrt(N) times when N = 2^n, n is number of input qubits
- Credit: https://qiskit.org/textbook/ch-algorithms/grover.html#5.-Solving-Sudoku-using-Grover's-Algorithm-

Quick Grover's Algorithm with 3 Qubits for example from Qiskit

https://qiskit.org/textbook/ch-algorithms/grover.html#3.-Example:-3-Qubits-

- 3 qubits with 2 marked states of |101> and |110>





1. Apply Hadamard gates to 3 qubits initialised to $|000\rangle$ to create a uniform superposition:

$$|\psi_1\rangle = \frac{1}{\sqrt{8}}(|000\rangle + |001\rangle + |010\rangle + |011\rangle + |100\rangle + |101\rangle + |110\rangle + |111\rangle)$$

2. Mark states $|101\rangle$ and $|110\rangle$ using a phase oracle:

$$|\psi_2\rangle = \frac{1}{\sqrt{8}}(|000\rangle + |001\rangle + |010\rangle + |011\rangle + |100\rangle - |101\rangle - |110\rangle + |111\rangle)$$

3. Perform the reflection around the average amplitude:
   1. Apply Hadamard gates to the qubits

   $$|\psi_{3a}\rangle = \frac{1}{2}(|000\rangle + |011\rangle + |100\rangle - |111\rangle)$$

   2. Apply X gates to the qubits

   $$|\psi_{3b}\rangle = \frac{1}{2}(-|000\rangle + |011\rangle + |100\rangle + |111\rangle)$$

   3. Apply a doubly controlled Z gate between the 1, 2 (controls) and 3 (target) qubits

   $$|\psi_{3c}\rangle = \frac{1}{2}(-|000\rangle + |011\rangle + |100\rangle - |111\rangle)$$

   4. Apply X gates to the qubits

   $$|\psi_{3d}\rangle = \frac{1}{2}(-|000\rangle + |011\rangle + |100\rangle - |111\rangle)$$

   5. Apply Hadamard gates to the qubits

   $$|\psi_{3e}\rangle = \frac{1}{\sqrt{2}}(-|101\rangle - |110\rangle)$$

4. Measure the 3 qubits to retrieve states $|101\rangle$ and $|110\rangle$

Note that since there are 2 solutions and 8 possibilities, we will only need to run one iteration (steps 2 & 3).

My Problem

```
# 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 |
# -----------------
```

Initial Implementation Setup

- Using the Qiskit Library and IBMQ Simulator
- Need to make an account with IBMQ to get access token
- Python 3.7.6 (Anaconda Environment) [Python 3.5+ required]
- Pip install qiskit

```
import qiskit as q
import matplotlib.pyplot as plt
from qiskit import IBMQ
import numpy as np
from qiskit.quantum_info.operators import Operator
```

Steps

1. Turn problem into a circuit
2. Create classical function that checks if a solution is valid
3. Identify rules/clauses
4. Iterate over clauses as the Oracle
5. Apply Phase Kickback to return from super position state
6. Apply the Diffuser and Oracle sqrt(N) times (22) = sqrt(512)
7. Measure input qubits

Turning the Problem into a circuit

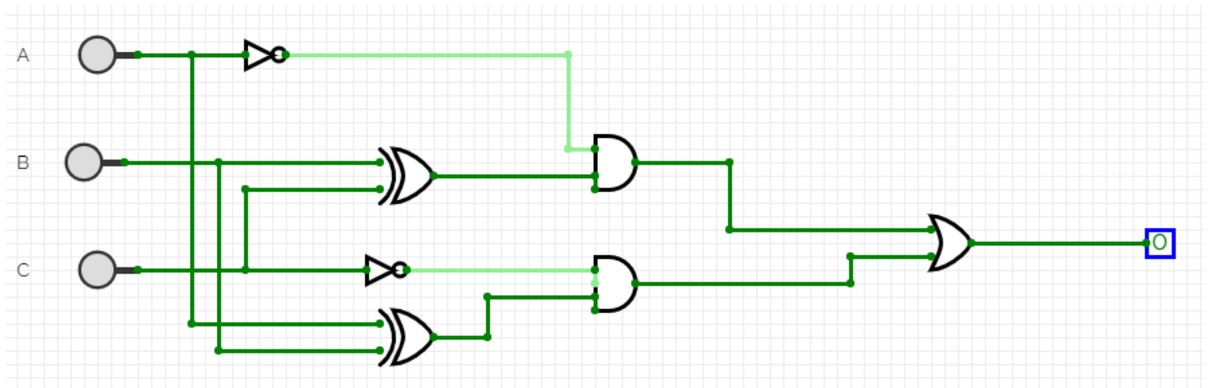
Identifying the Clauses or Rules of the Game

```
# To do this we first need to define the conditions to check:
# for each row/column the solutions can boil down to
#  ((NOT A) AND (B XOR C) OR ((NOT C) AND (A XOR B ))
# 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]]
```

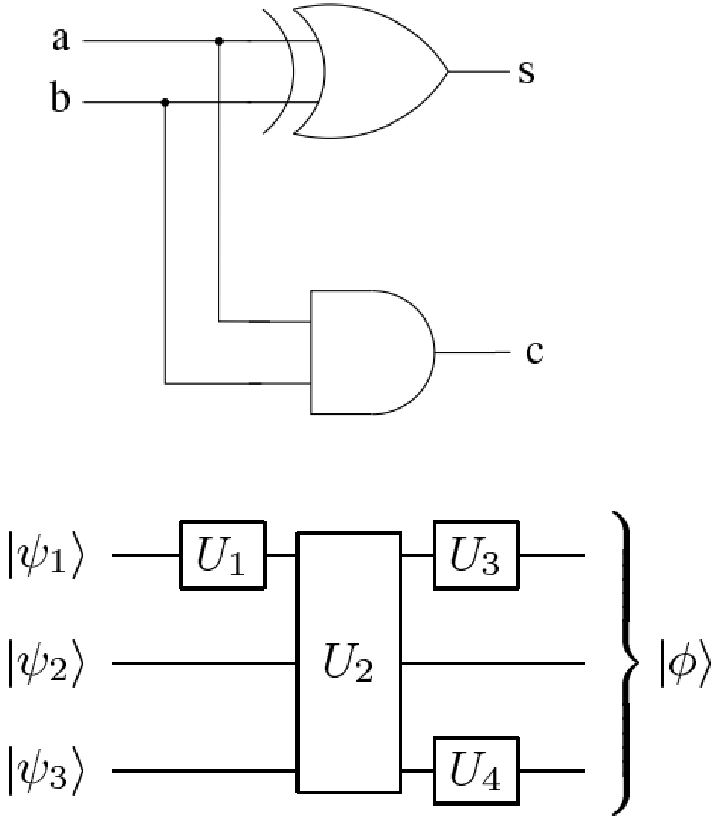
```
-----------------
| vo | v1 | v2 |
-----------------
| v3 | v4 | v5 |
-----------------
| v6 | v7 | v8 |
-----------------
```

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



Quantum Circuits vs Classical Circuits

- 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 states to different points in the Circuit
- Example of half-adder in classical and random quantum circuit.
- Credit: https://arxiv.org/pdf/1110.2998.pdf



Classical Logical Half-Adder (top) and Generic Quantum Circuit (bottom)

Classical Gates as Quantum Gates

```
# The final output should be the Quantum Equivalent of Classical:
# ((NOT A) AND (B XOR C) OR ((NOT C) AND (A XOR B ))

# The NOT Gate in classical can be represented by an X Gate (NOT)
# The XOR Gate can be represented by the combination of 2 CX (CNOT)
# The AND Gate can be implemented using a Toffoli Gate ccx (CCNOT)

# No current quantum gates are explicit creations of an OR Gate
# because all operations must be reversible.
```

Credit: https://qiskit.org/textbook/ch-states/atoms-computation.html

X Gate (NOT)

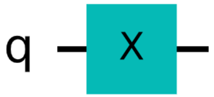
1.1 The X-Gate

The X-gate is represented by the Pauli-X matrix:

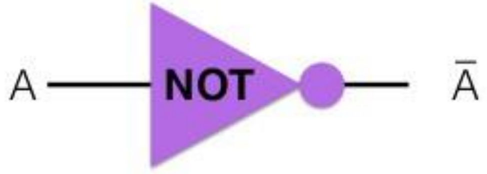
$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = |0\rangle\langle 1| + |1\rangle\langle 0|$$

```
# Let's do an X-gate on a |0> qubit
qc = QuantumCircuit(1)
qc.x(0)
qc.draw()
```

try

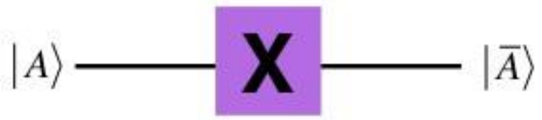


https://qiskit.org/textbook/ch-states/single-qubit-gates.html#xgate



A	Ā
0	1
1	0

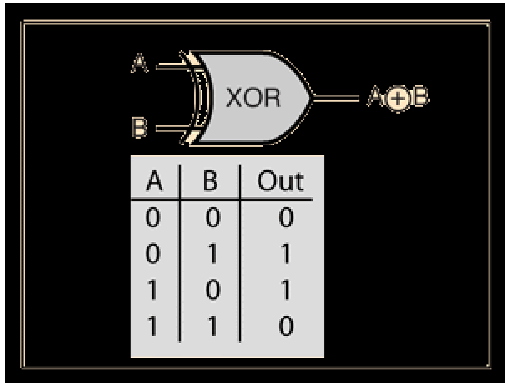
PAULI X GATE



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

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

CNOT (XOR)



http://hyperphysics.phy-astr.gsu.edu/hbase/Electronic/xor.html

When our qubits are not in superposition of $|0\rangle$ or $|1\rangle$ (behaving as classical bits), this gate is very simple and intuitive to understand. We can use the classical truth table:

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:

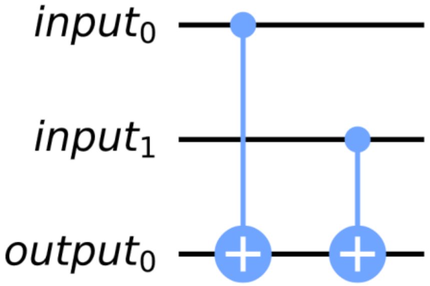
$$\text{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}, \quad \text{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

https://qiskit.org/textbook/ch-gates/multiple-qubits-entangled-states.html#cnot

```python
def XOR(qc, a, b, output):
    qc.cx(a, output)
    qc.cx(b, output)
```

```python
# We will use separate registers to name the bits
in_qubits = QuantumRegister(2, name='input')
out_qubit = QuantumRegister(1, name='output')
qc = QuantumCircuit(in_qubits, out_qubit)
XOR(qc, in_qubits[0], in_qubits[1], out_qubit)
qc.draw()
```

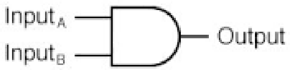
try



https://qiskit.org/textbook/ch-algorithms/grover.html#5.-Solving-Sudoku-using-Grover's-Algorithm-

Toffoli (AND)

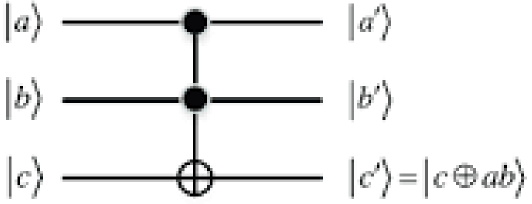
2 - input AND gate



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

https://www.allaboutcircuits.com/textbook/digital/chpt-3/multiple-input-gates/

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



https://www.researchgate.net/figure/Truth-table-and-quantum-circuit-of-Toffoli-gate_fig3_338021358

Circuit symbol:

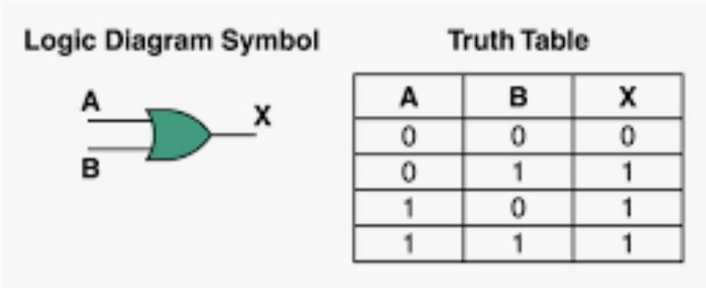
```
q_0: ──■──
       │
q_1: ──■──
     ┌─┴─┐
q_2: ┤ X ├
     └───┘
```

Matrix representation:

$$CCXq_0, q_1, q_2 = I \otimes I \otimes |0\rangle\langle 0| + CX \otimes |1\rangle\langle 1| = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

https://qiskit.org/documentation/stubs/qiskit.circuit.library.CCXGate.html#qiskit.circuit.library.CCXGate

Creating the OR Operator



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

Credit: https://quantumcomputing.stackexchange.com/a/5834

```
# To create the OR Logic, we need to use a unitary matrix using the Operator
# class and use an Ancilla Bit (extra bit)
# Ancilla Bit - https://en.wikipedia.org/wiki/Ancilla_bit

OR_Unitary_Matrix = [[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]
                     ]
```

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

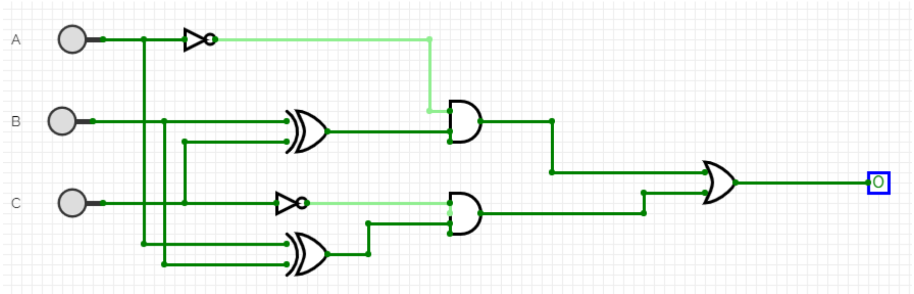
Input			Output		
0	0	1	0	0	0
0	1	1	0	1	1
1	0	1	1	0	1
1	1	1	1	1	1

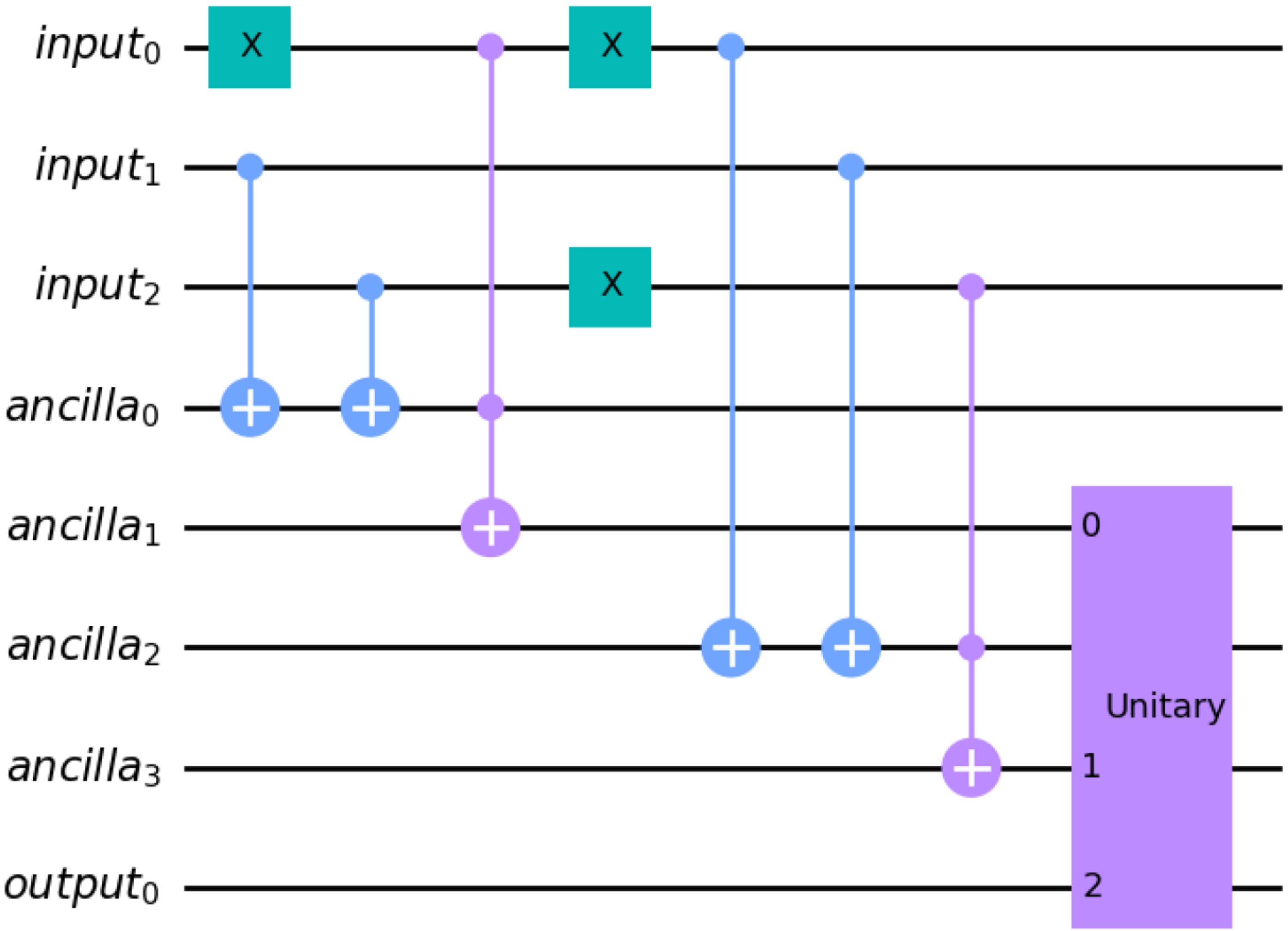
Input			Output		
0	0	0	0	0	1
0	0	1	0	0	0
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	1	0	1
1	1	0	1	1	0
1	1	1	1	1	1

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

```
# I tested this matrix to make sure it truly is unitary using the Qiskit is_unitary() function.
```

Quantum Equivalent of Classical: ((NOT A) AND (B XOR C) OR ((NOT C) AND (A XOR B))

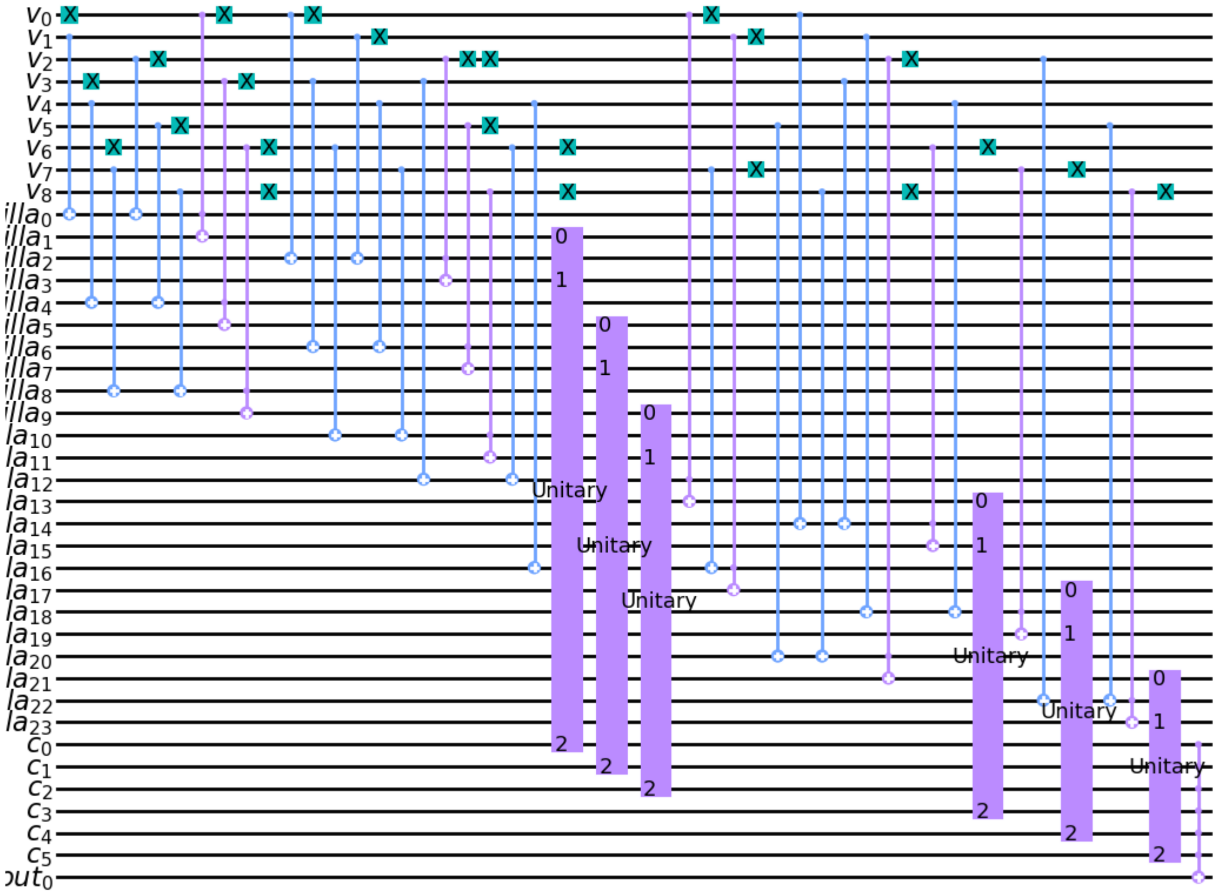




```python
def OnlyOneTrue(qc, a, b, c,  ancilla_2, ancilla_3, out_qubit):
    # NOT A
    qc.x(a)
    # B XOR C
    # ancilla_1 will be flipped only if b and c are different.
    qc.cx(b, out_qubit)
    qc.cx(c, out_qubit)
    # NOT A AND (B XOR C) - stored in ancilla_2
    qc.ccx(a, out_qubit, ancilla_2)
    # RESET ancilla_1 will be flipped back only if b and c are different.
    qc.cx(b, out_qubit)
    qc.cx(c, out_qubit)
    # Need to undo NOT A for second half.
    qc.x(a)
    # NOT C
    qc.x(c)
    # A XOR B
    # ancilla_1 will be flipped only if b and a are different.
    qc.cx(a, out_qubit)
    qc.cx(b, out_qubit)
    # NOT C AND (B XOR A) - stored in ancilla_3
    qc.ccx(c, out_qubit, ancilla_3)
    # RESET ancilla_1 will be flipped only if b and a are different.
    qc.cx(a, out_qubit)
    qc.cx(b, out_qubit)
    # OR Operation
    qc.append(OR_Operator, [ancilla_2, ancilla_3, out_qubit])
    qc.x(c)  # Reset Value of C
```

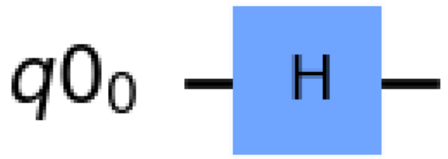
Creation/Testing (Oracle)

- The Quantum Circuit is created twice so that any changes to the initial Qubits will be reversed so that they can be measured at the end of the algorithm.
- This is one iteration of the circuit with all 6 clauses and 9 variable qubits, 24 extra, 7 outputs. (need to reduce this to simulate under 32 qubits)



Putting it together + Hadamard Gate

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




```
# Create separate registers to name bits
var_qubits = q.QuantumRegister(9, name='v')
ancilla_qubits = q.QuantumRegister(12, name='ancilla')
clause_qubits = q.QuantumRegister(6, name='c')
output_qubit = q.QuantumRegister(1, name='out')
cbits = q.ClassicalRegister(9, name='cbits')

# Create Quantum Circuit
qc = q.QuantumCircuit(var_qubits, ancilla_qubits,
                      clause_qubits, output_qubit, cbits)

# Initialize 'out0' in state |->
qc.initialize([1, -1]/np.sqrt(2), output_qubit)

# Initialize qubits in state |s>
qc.h(var_qubits)
qc.barrier()  # For Visual Separation (may remove later)
```

```
# Next, create a checking circuit using the Oracle using phase kickback
def sudoku_oracle(qc, clause_list, clause_qubits):
    # Use OnlyOneTrue gate to check each clause
    i = 0
    for clause in clause_list:
        OnlyOneTrue(qc, clause[0], clause[1], clause[2], ancilla_qubits[2*i],
                    ancilla_qubits[2*i + 1],
                    clause_qubits[i])
        i += 1
    # Flip 'output' bit if all classes are satisfied
    qc.mct(clause_qubits, output_qubit)
    # Uncompute clauses to reset clause-checking bits to 0
    i = 0
    for clause in clause_list:
        OnlyOneTrue(qc, clause[0], clause[1], clause[2], ancilla_qubits[2*i],
                    ancilla_qubits[2*i + 1],
                    clause_qubits[i])
        i += 1
```

Full Grover's Algorithm

```python
# Here we need 22 iterations since there are 2^9 combinations and s

for i in range(22):
    # Apply our oracle
    sudoku_oracle(qc, clause_list, clause_qubits)
    qc.barrier()  # For Visual Separation (may remove later)
    # Apply our diffuser
    qc.append(diffuser(9), [0, 1, 2, 3, 4, 5, 6, 7, 8])

# Measure the variable qubits
qc.measure(var_qubits, cbits)
```

```python
# Simulate the Circuit using the open-source QASM Simulator provided by IBM Q.
provider = IBMQ.load_account()
backend = provider.get_backend('ibmq_qasm_simulator')
transpiled = q.transpile(qc, backend=backend)
qobj = q.assemble(transpiled)
job = backend.run(qobj)
retrieved_job = backend.retrieve_job(job.job_id())
result = job.result()
counts = result.get_counts()
print(counts)

# Used for showing Circuits and Histograms.
plt.show()
```

Simulation and Plot Results

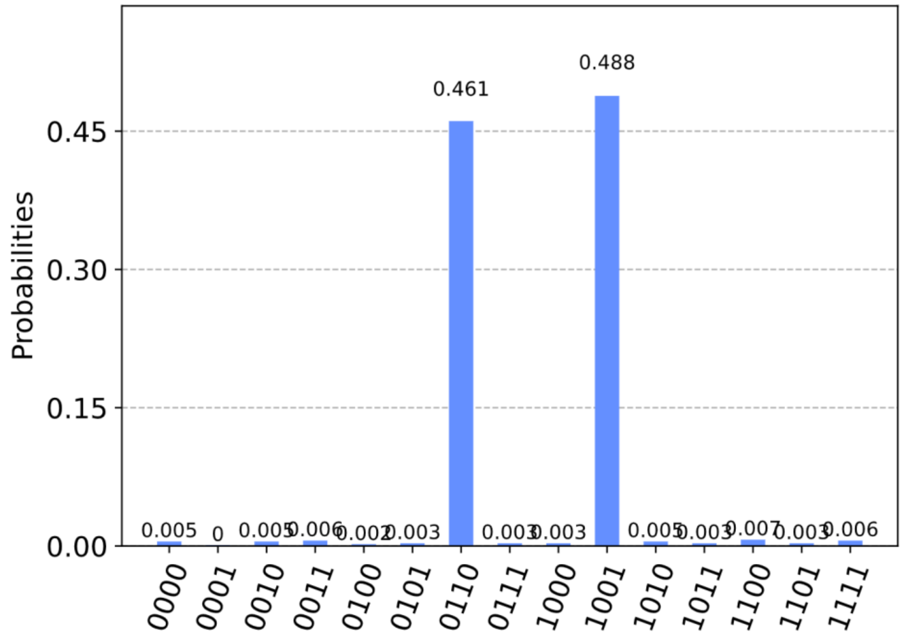
V_0	V_1
V_2	V_3

- Expected Results for 2 by 2.
- CHANGES expected for 3 by 3.
- Total of 6 Solutions, so each of the probabilities would be about 3/256.
- 2^9 512 total combinations instead of 2^4 = 16
- 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 |
-----------------
```

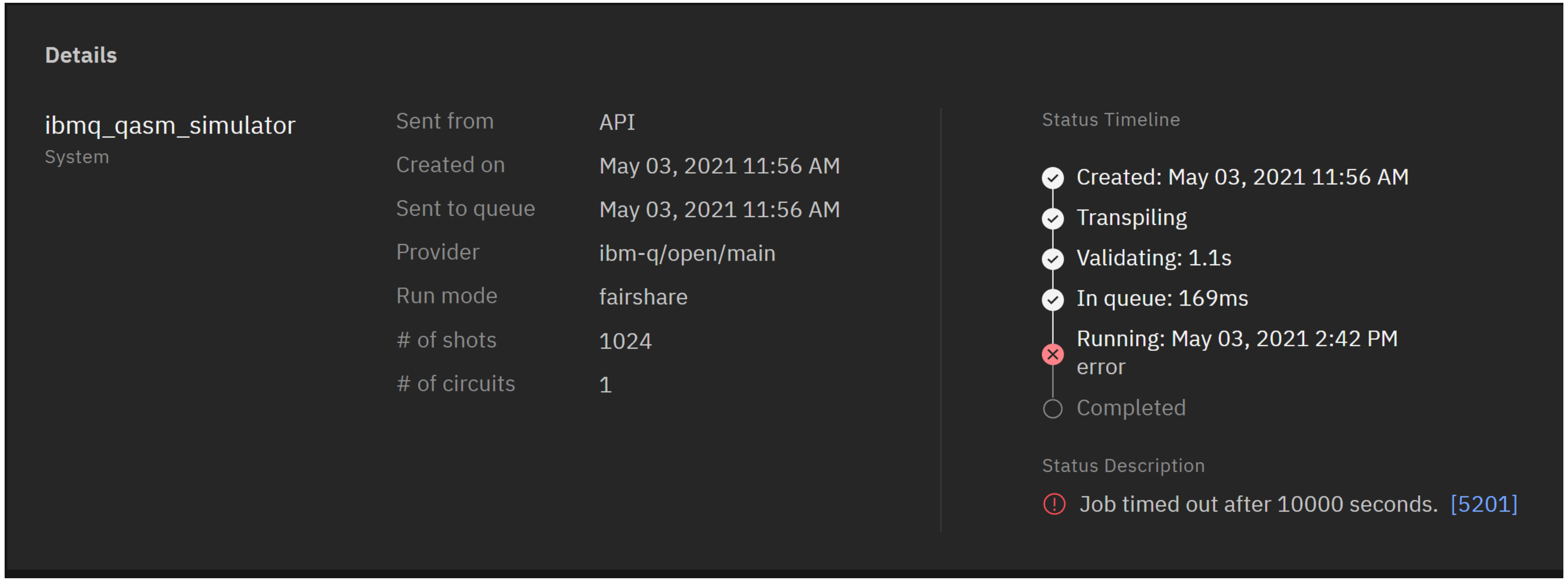


https://qiskit.org/textbook/ch-algorithms/grover.html#5.3-The-Full-Algorithm

Practical Limitations

- Limitations on number of Qubits able to use with IBMQ_QASM_Simulator at 32 Qubits

https://quantumcomputing.stackexchange.com/questions/14927/maximum-number-of-qubits-suppoted-by-the-qasm-simulator



Further Applications

A quick Google search on Applications of Grover's Algorithm shows many problems that Grover's Algorithm can be applied to that significantly decreases search time.

Examples:

Collision Problem - https://en.wikipedia.org/wiki/Collision_problem

Polynomial Root Finding Problem - https://ieeexplore.ieee.org/document/7016940

Transcendental Logarithm Problem - https://ieeexplore.ieee.org/document/7016935

References

- Link to GitHub Repository with my code https://github.com/theRealNoah/qiskit-grovers-3by3/blob/main/grovers_3by3.py

Other Resources used to create and learn about Quantum Circuits/Algorithms.

- https://qiskit.org/textbook/ch-algorithms/grover.html
- Borbely, E., "Grover search algorithm", *arXiv e-prints*, 2007.
- Chapter 6 Quantum Algorithms – Professor Chen USF
- https://en.wikipedia.org/wiki/Grover%27s_algorithm
- Garcia-Escartin J.C., Charmorro-Posada P. "Equivalent Quantum Circuits", *arXiv:1110.2998v1* [quant-ph] 13 Oct 2011. https://arxiv.org/pdf/1110.2998.pdf
- https://qiskit.org/textbook/ch-states/atoms-computation.html
- https://towardsdatascience.com/demystifying-quantum-gates-one-qubit-at-a-time-54404ed80640
- IBM Quantum. https://quantum-computing.ibm.com/, 2021