Quantum Algorithms

Nicholas “Nick” Allgood

Who’s this guy?

PhD Student, Computer Science. Candidacy anticipated early 2021.

Masters Thesis Defense: April 28 (Using quantum algorithms for malware analysis).

Day Job: Self-employed, work full time as consultant. Primarily as a network architect, but I do a myriad of things in my day job.

Research Interests: Quantum Computing (teleportation, communication, cryptography, cybersecurity), Classical communication networks and cybersecurity.

Agenda

Part 1:	Brief Review Quantum Computing
Part 2:	Deutsch-Jozsa
Part 3:	Simon’s Problem / Algorithm / Current Research
Part 4:	Grover’s Algorithm / Current Research
Part 5:	Questions / Comments / Complaints

Part 1: Brief Review of Quantum Computing

Types of Quantum Computers

Adiabatic (Quantum Annealer): Used to find the minimum global state and is the more common of quantum computers seen today. Typically useful for optimization problems (NP-Hard) [5].

Analog (Quantum Simulator): These are used to simulate complex quantum interactions and able to detect untraceable quantum details. Most likely will start to see dramatic improvements over classical computing [5].

Universal: The “holy grail” of quantum computing that combines the full power of classical and quantum computation and will be the most general purpose quantum computer [5].

Hybrid: Combines quantum annealing and classical computing together in one machine, but only uses quantum CPU (QPU) for certain operations [6].

Measurement Based: Starts with a fixed entangled state and then a sequence of measurements [16].

Qubits!

A quantum bit or qubit is the quantum equivalent of a classical Shannon bit.

A classical bit is a boolean value represented by a 1 or 0 (True or False)

A qubit is a superposition of 0 and 1 at the same time where one number tells you the probability of being 0 and one tells you the probability of being a 1*.

100% chance of being 0, 0% chance of 1

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

0% chance of being 0, 100% chance of 1

In quantum mechanics, you will often see "0" being -1 and "1" being 1

Bra vs kets

A ket is used to represent a vector space as a column.

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

A bra is used to represent a vector space as a row.

$$\langle 0| = \begin{pmatrix} 1 & 0 \end{pmatrix}$$

While it may seem like a bra and ket are the same, they can have different uses individually -- hence why they are separate.

Qubits and Brakets

Ket’s and bra’s are used to represent a single qubit (though ket’s are the most common).

A single qubit is nothing but a vector space, as such we can start to do some math on them. We do an tensor (Kronecker) product on two qubits and we get a larger vector space**.

$$|0\rangle \otimes \langle 0| = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$$

The result expands the size of our quantum system exponentially, in addition provides us with the resulting quantum state of the system.

***Literature often implies and omits the tensor symbol.*

$$|0\rangle \langle 0| = |0\rangle \otimes \langle 0|$$

Quantum Registers

Like classical computing, quantum computers have registers known as quantum registers.

These registers are made up of multiple qubits and calculations are done by manipulating the qubits within a quantum register.

Remember that qubits are represented by a vector space, so that means multiple qubits put together (tensored) make a quantum register. The quantum register is then represented by the resulting sized vector.

Quantum Logic Gates

Similar to classical computing, quantum computing also has logic gates known as quantum logic gates or quantum gates.

Quantum gates are a foundational part of a universal quantum computer.

Represented as a unitary matrix, which is a special type of square matrix.

Many gates are single qubit gates, but there are some that operate on more than one qubit (which then causes quantum entanglement).

Common Quantum Gates

Pauli Gates

X (NOT)

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

Y (Rotate Y-Axis)

$$Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$$

Z (Rotate Z-Axis / Phase Flip)

$$Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

Hadamard (Superposition)

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

SWAP

$$\mathrm{SWAP} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

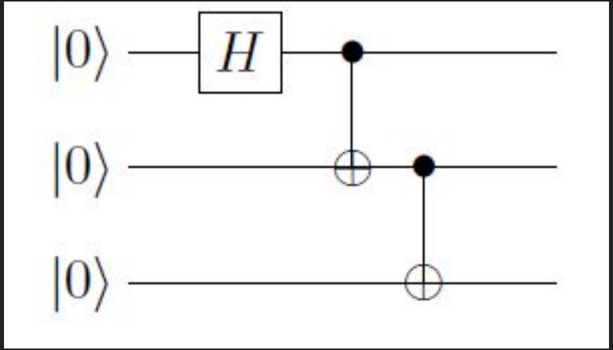
CNOT

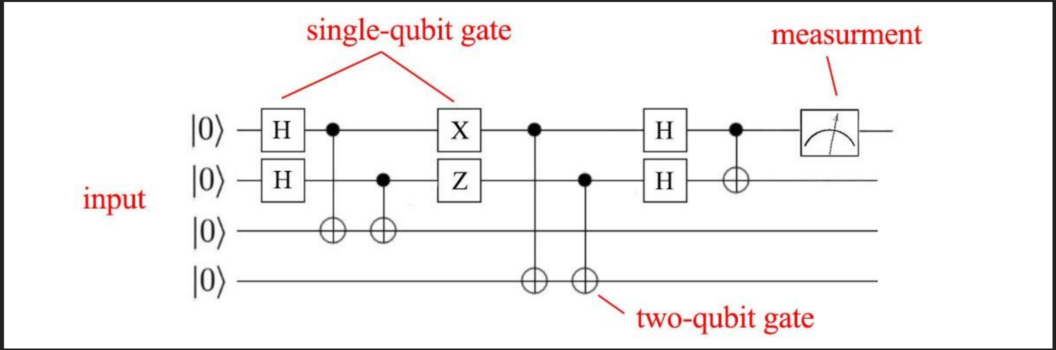
$$\mathrm{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Quantum Circuit

Quantum gates are used to build a quantum circuit, much in the same fashion that classical logic gates are used to build a logic circuit.

Examples:





*Original image
https://medium.com/@jonathan_hui/qc-programming-with-quantum-gates-8996b667d256

Quantum Systems and Quantum States

A quantum system “is a portion of the whole universe taken to analyze the wave-particle duality in that system [3].” A quantum state is a particular state of everything going on in that system at that present time (similar to a snapshot).

A quantum computer is a quantum system and its state is a ‘snapshot’ of the all the qubit data and linear combinations being performed on that data.

You will see a lot of references to energy and amplitudes when reading about quantum computing...this is the quantum mechanics coming through...with quantum computing almost being a very thin layer on top.

Quantum Measurement

Quantum measurement is when we measure the state of a quantum system and is measured using an observable (which is represented as a matrix)

Once the measured state is returned, we also use that information to determine the probability of getting that state.

Quantum Entanglement

Quantum entanglement is where a particle interacts with another particle and one can’t tell their quantum states apart.

In quantum computing, this is done when we have a multi-qubit gate applied to at least two qubits.

Once applied, the original state of each of the original qubits can no longer be determined.

I.e. The CNOT gate is a gate that causes entanglement on two qubits.

Noise in Quantum Systems

Two types: Classical and Quantum.

Classical noise is typically due to known factors such as vibrations, variations thermal energy in a laser emitter, but could be also due to unknown reasons.

Quantum noise is known as decoherence which simply put is the loss of quantum behavior.

While quantum physicists are interested in studying noise, computer scientists wish to reduce the noise as much as possible (to get the most accurate result).

Part 2: Deutsch-Jozsa

Foundational Algorithm

Deutsch-Jozsa is one of the foundational algorithms in quantum computing.

Inspired many other algorithms, including Simon’s algorithm, Grover’s algorithm, etc.

Introduces oracle or “blackbox” computing.

Simply put, an oracle or blackbox is some entity that takes inputs and gives you an expected output, however you don’t necessarily know how it works on the inside.

The Blackbox Problem

The blackbox concept is carried over from quantum physics. Quantum physicists do not care about the inner workings of a blackbox (typically).

As computer scientists, we **DO** care, since we have to implement the oracle!

The problem is since quantum computing is still fairly early in development, a lot of literature references “some oracle” but doesn’t really talk about how to make the oracle work -- you are left to your own devices most of the time.

At a very high level, you can think of it almost like an API function. You know what the function should take as input parameters and the output, but not necessarily how it works internally.

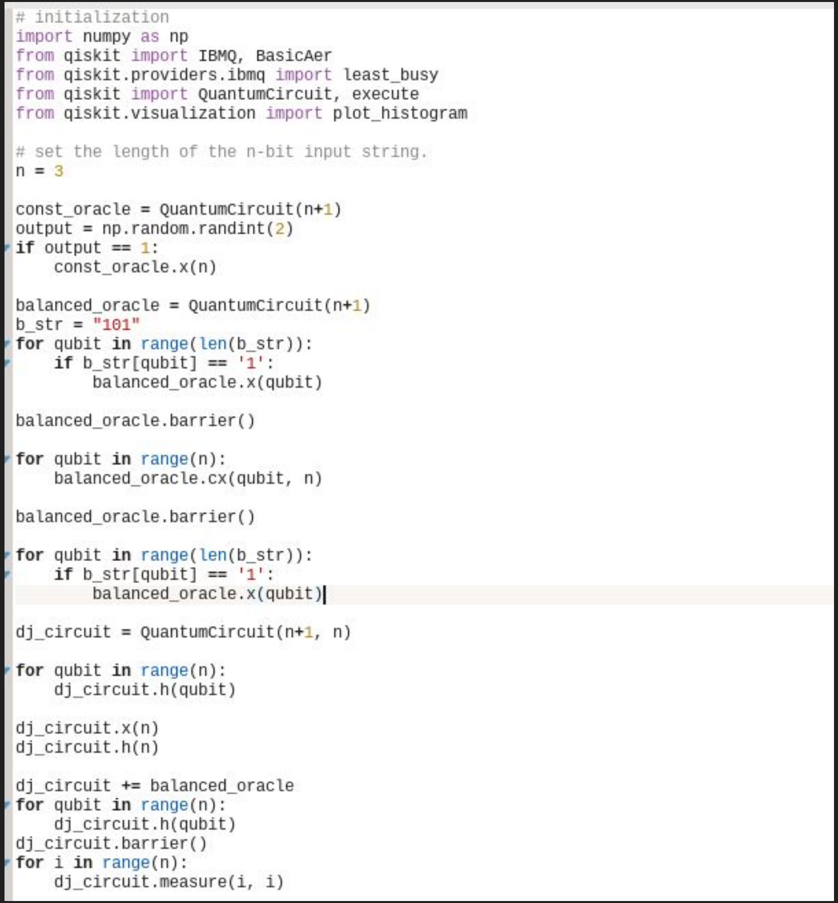
Deutsch-Jozsa’s Algorithm

Created by David Deutsch and Richard Jozsa in 1992.

Formal Definition: Given a blackbox quantum oracle that implements some function $f : \{0, 1\} \mapsto \{0, 1\}$. The function takes an n-digit binary value as the input and produces either 0 or 1 as the output for each value. The function promises that if all outputs returned are 0, the function is constant. If we get at least 1 in the output (or all outputs), then the function is balanced (half the inputs return 1, the other half returns 0).

The oracle simply returns whether our function is balanced (0), or constant (1).

Deutsch-Jozsa Example (Python - Qiskit)



```python
# initialization
import numpy as np
from qiskit import IBMQ, BasicAer
from qiskit.providers.ibmq import least_busy
from qiskit import QuantumCircuit, execute
from qiskit.visualization import plot_histogram

# set the length of the n-bit input string.
n = 3

const_oracle = QuantumCircuit(n+1)
output = np.random.randint(2)
if output == 1:
    const_oracle.x(n)

balanced_oracle = QuantumCircuit(n+1)
b_str = "101"
for qubit in range(len(b_str)):
    if b_str[qubit] == '1':
        balanced_oracle.x(qubit)

balanced_oracle.barrier()

for qubit in range(n):
    balanced_oracle.cx(qubit, n)

balanced_oracle.barrier()

for qubit in range(len(b_str)):
    if b_str[qubit] == '1':
        balanced_oracle.x(qubit)

dj_circuit = QuantumCircuit(n+1, n)

for qubit in range(n):
    dj_circuit.h(qubit)

dj_circuit.x(n)
dj_circuit.h(n)

dj_circuit += balanced_oracle
for qubit in range(n):
    dj_circuit.h(qubit)
dj_circuit.barrier()
for i in range(n):
    dj_circuit.measure(i, i)
```

*From https://qiskit.org/textbook/ch-algorithms/deutsch-josza.html#djalgorithm

Part 3: Simon’s Problem

Simon's Algorithm - Formal

Formal Definition: For a given function (using a blackbox or oracle),

$f(x) : \{0,1\}^n \mapsto \{0,1\}^n$, promises to satisfy the property for some $s \in \{0,1\}^n$

$$\forall\, x, y \in \{0,1\}^n, f(x) = f(y) \iff x \oplus y \in \{0^n, s\}$$

In the case of s = 0, then f is required to be a one-to-one function (two-to-one otherwise).

f is a function such that, $f(x) = f(x \oplus y), \forall\, x \in \{0,1\}^n$ and given a fixed and unknown $s \in \{0,1\}^n$

Simon’s Algorithm - Layperson

We have a blackbox/oracle that will take $|x\rangle|y\rangle$ and produce $|x\rangle|f(x) \oplus y\rangle$ for an input bitstring with a hidden bitstring (s) and map to two outputs.

We wish to find this hidden bitstring. (Often called a *period* string in literature).

Simon’s algorithm will return a series of linear equations that can then be solved classically to find the hidden period string.

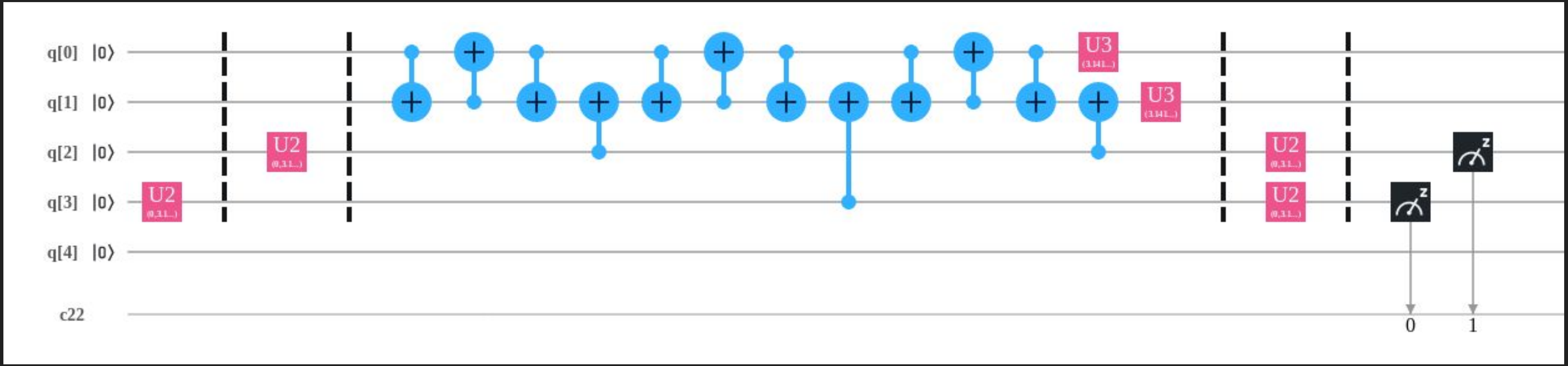
Practical Uses

Currently has been used for research to benchmark the “quantumness” of the IBM Q quantum computers. (Allgood, Aborle, Lomanoco) - Scheduled for IEEE submission by May 8 and hopefully accepted by late summer. Conference tentatively in Denver, CO in October.

The purpose is to determine how often the IBMQ computers give us a correct result to Simon’s problem. Simulators will always give correct results due to lack of noise.

Since Simon’s utilizes a 2-to-1 function, many 2-to-1 functions are used heavily in cryptography and encryption, so there are potential research ideas there.

Example Simon’s Quantum Circuit



** Circuit taken from Simon’s Algorithm 2-bit string using 5-qubit backend ibmq_burlington

Results - 5 Qubit

Each circuit is ran 1024 times on the quantum backend.

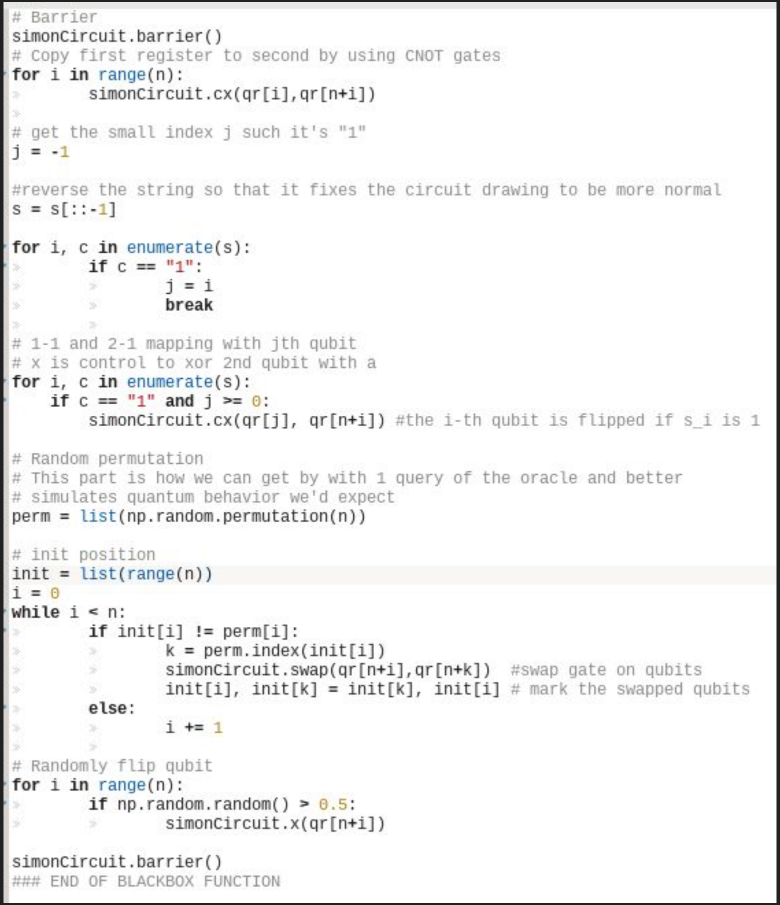
Backend	Qubits	Bitstring Length	Total Runs	Correct	Percent Correct	Incorrect	Percent Incorrect
Ibmqx2 (Yorktown)	5	2	4096	3338	81.494%	758	18.505%
london	5	2	4096	2537	61.938%	1559	38.061%
ourense	5	2	4096	3076	75.097%	1020	24.902%
essex	5	2	4096	3045	83.129%	691	16.870%
burlington	5	2	4096	3049	74.438%	1047	25.561%
vigo	5	2	4096	3325	81.176%	771	18.823%

Results - 14 Qubit

7-bit strings is 1024 runs * 63 unique bit strings. 2-7 bit strings are 1024 runs * 139 unique bit strings.

Backend	Qubits	Bitstring Length	Total Runs	Correct	Percent Correct	Incorrect	Percent Incorrect
16_melbourne	14	2	4096	3008	73.473%	1088	26.562%
16_melbourne	14	7	64512	32572	50.048%	31940	49.510%
16_melbourne	14	2 - 7	142336	73733	51.802%	68603	48.197%

Sample Blackbox (Qiskit - Python)



```
# Barrier
simonCircuit.barrier()
# Copy first register to second by using CNOT gates
for i in range(n):
        simonCircuit.cx(qr[i],qr[n+i])

# get the small index j such it's "1"
j = -1

#reverse the string so that it fixes the circuit drawing to be more normal
s = s[::-1]

for i, c in enumerate(s):
        if c == "1":
                j = i
                break

# 1-1 and 2-1 mapping with jth qubit
# x is control to xor 2nd qubit with a
for i, c in enumerate(s):
    if c == "1" and j >= 0:
        simonCircuit.cx(qr[j], qr[n+i]) #the i-th qubit is flipped if s_i is 1

# Random permutation
# This part is how we can get by with 1 query of the oracle and better
# simulates quantum behavior we'd expect
perm = list(np.random.permutation(n))

# init position
init = list(range(n))
i = 0
while i < n:
        if init[i] != perm[i]:
                k = perm.index(init[i])
                simonCircuit.swap(qr[n+i],qr[n+k])  #swap gate on qubits
                init[i], init[k] = init[k], init[i] # mark the swapped qubits
        else:
                i += 1

# Randomly flip qubit
for i in range(n):
        if np.random.random() > 0.5:
                simonCircuit.x(qr[n+i])

simonCircuit.barrier()
### END OF BLACKBOX FUNCTION
```

Sample Output

```
IBM Q Backend ibmqx2: Resulting Values and Probabilities

===============================================

Simulated Runs: 1024

Period: 01 , Counts: 196 , Probability: 0.19140625

Period: 11 , Counts: 297 , Probability: 0.2900390625

Period: 10 , Counts: 269 , Probability: 0.2626953125

Period: 00 , Counts: 262 , Probability: 0.255859375
```

Part 4: Grover’s Algorithm

Grover’s Search

One of the first quantum searching algorithms created by Lov K. Grover

Inspiration for Shor’s factoring algorithm.

Has since been improved vastly over the years and widely researched.

Only more efficient on unsorted and unstructured data.

Makes heavy “use” and reference of the Bloch Sphere.

Bloch Sphere

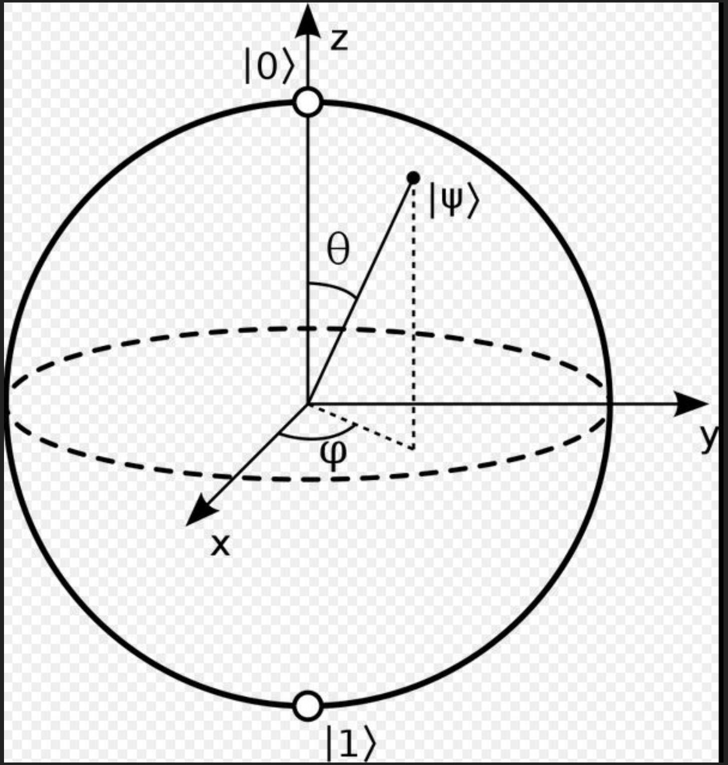
The Bloch Sphere is a geometrical representation of a two-level quantum mechanical system (qubits).

When thinking about quantum computing, it’s very helpful to utilize the Bloch Sphere as a model.

The Bloch sphere is also an intuitive way to look at particles and their respective spins.

We have several states which correspond to the Pauli gates mentioned earlier. (X, Y, Z). We have “spin-up” , “spin-down”, “diagonal spin”, “rotations” etc

Bloch Sphere (Drawing)



* https://en.wikipedia.org/wiki/Bloch_sphere#/media/File:Bloch_sphere.svg

Amplitude Amplification

Grover’s utilizes amplitude amplification initially created by Gilles Brassard in 1997 and later that year by Grover.

In fact, amplitude amplification is more or less a generalization of Grover’s Algorithm.

The fundamental idea is to increase (amplify) the probabilities of the desired results, and this is accomplished by using a sequence of reflections.

Amplitude amplification causes the reflections to be rotated closer to the desired quantum state along the Bloch Sphere.

The target state is marked as $\sin^2(\Theta)$ and the algorithm is applied m-times.

The probability of obtaining the correct state is $\sin^2((2m+1)\Theta)$

Simply put, we have a target state on a sphere, we keep rotating it until we find the correct result. Each rotation brings us slightly closer.

Phase Amplification

With phase amplification, we do a variety of operations to change the phase of our amplitude that matches our target value.

Before we get to searching for our target value, we have to change our thinking when it comes to searching a lookup table.

Typically we want to get a known key and then we can look up its corresponding value.

With Grover’s, we flip this idea upside down, we know the value we are looking for , but we don’t actually know the key.

Phase Amplification Example

For practical purposes, in quantum computing phase amplification is associated with changing the sign of a value from between + or -.

For example, say we look at the following matrix and we wish to locate the value at row 1, column 3:

$$\begin{bmatrix} AB & CD & \mathbf{EF} \\ 12 & 97 & 85 \\ 2D & 3F & 9C \end{bmatrix} \xmapsto{\text{Phase Flip}} \begin{bmatrix} AB & CD & -\mathbf{EF} \\ 12 & 97 & 85 \\ 2D & 3F & 9C \end{bmatrix}$$

Entanglement in Action

Grover’s makes heavy use of quantum entanglement.

Lookup table is loaded into a quantum machine and Grover’s algorithm will entangle all permutations of potential key and value pairs.

The next piece is to do phase amplification by placing a *tag* value that equals our search value.

Ties heavily into the quantum fourier transform.

Represented in a quantum circuit by a Pauli-Z gate or a Control-Z (CZ) gate.

Asymptotics

Grover’s search only takes $O(\sqrt{N})$ number of lookups on real quantum hardware.

With simulators, there a caveat where it is much slower due to having to visit each of the permutations with every iteration. This is an implementation only issue, NOT an algorithmic one.

Classically, a linear value search would take $O(N)$ number of lookups.

One could also potentially reverse the lookup table, but provided it’s a single-linked list, this would take $O(N) + O(1) = O(N)$

Practical Use

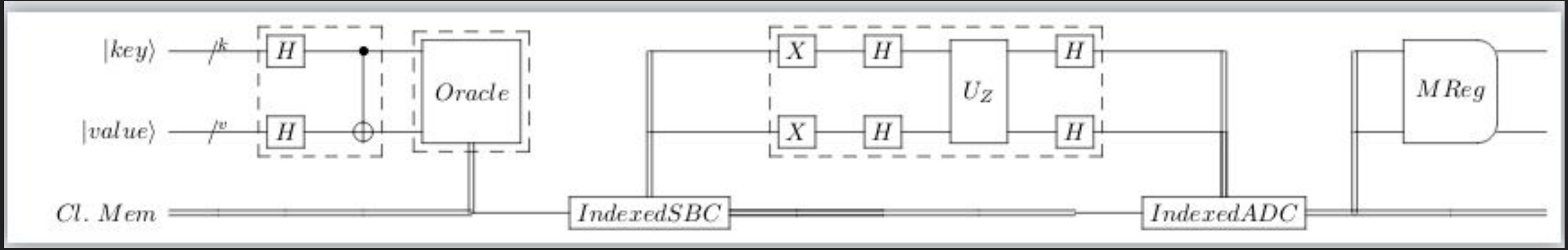
Making heavy use for my masters thesis topic, and will also be another IEEE paper submitted in May (Allgood, N. and Nicholas, C.)

Ngrams are used in many things, but they can be very useful in malware analysis and research.

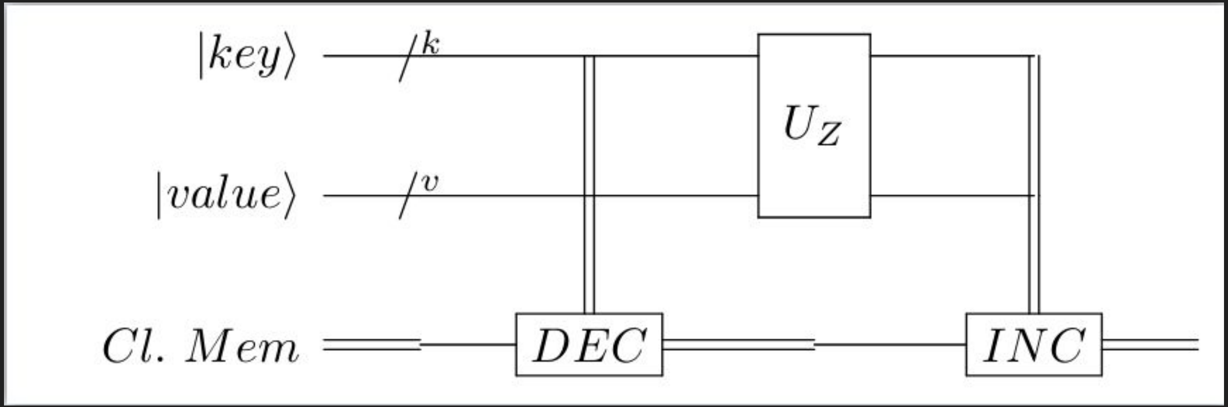
We are given a generated hash table from a malicious corpus and need to obtain a hash associated with a known malicious byte string.

This prevents us from having to iterate through a list of byte-strings and re-hash them, which also takes $O(MN)$.

Grover’s Circuits



Grover’s Circuit



Grover’s Oracle

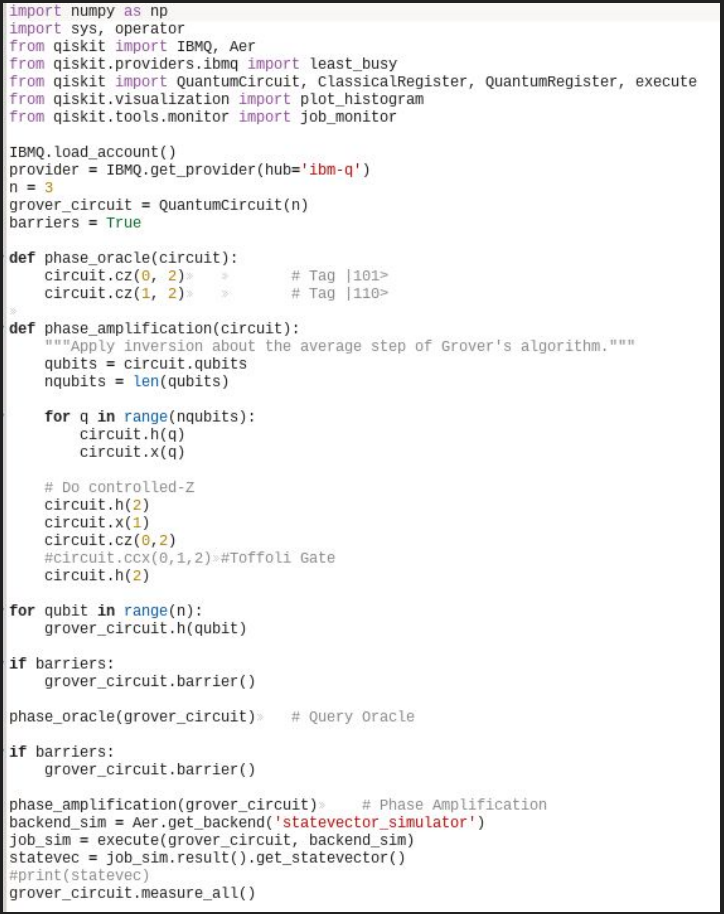
Grover's Example (Python - Qiskit)

We "tag" two input states:

$$|110\rangle, |101\rangle$$

The "quantum endianness" for IBMQ machines is MSB to LSB.

In this example the left most qubit is labeled "2" in the code while the right most qubit is "0" in the code, with "1" being in the middle.



```python
import numpy as np
import sys, operator
from qiskit import IBMQ, Aer
from qiskit.providers.ibmq import least_busy
from qiskit import QuantumCircuit, ClassicalRegister, QuantumRegister, execute
from qiskit.visualization import plot_histogram
from qiskit.tools.monitor import job_monitor

IBMQ.load_account()
provider = IBMQ.get_provider(hub='ibm-q')
n = 3
grover_circuit = QuantumCircuit(n)
barriers = True

def phase_oracle(circuit):
    circuit.cz(0, 2)            # Tag |101>
    circuit.cz(1, 2)            # Tag |110>

def phase_amplification(circuit):
    """Apply inversion about the average step of Grover's algorithm."""
    qubits = circuit.qubits
    nqubits = len(qubits)

    for q in range(nqubits):
        circuit.h(q)
        circuit.x(q)

    # Do controlled-Z
    circuit.h(2)
    circuit.x(1)
    circuit.cz(0,2)
    #circuit.ccx(0,1,2) #Toffoli Gate
    circuit.h(2)

for qubit in range(n):
    grover_circuit.h(qubit)

if barriers:
    grover_circuit.barrier()

phase_oracle(grover_circuit)    # Query Oracle

if barriers:
    grover_circuit.barrier()

phase_amplification(grover_circuit)     # Phase Amplification
backend_sim = Aer.get_backend('statevector_simulator')
job_sim = execute(grover_circuit, backend_sim)
statevec = job_sim.result().get_statevector()
#print(statevec)
grover_circuit.measure_all()
```

*https://qiskit.org/textbook/ch-algorithms/grover.html

Note the code on the left was modified from the original example since it appeared the original example contained errors in either the tag values they wanted to use, or the phase amplification gates.

Also note in simulation you should typically get the target value(s) you are looking for with an equal probability distribution.

8 Possibilities, 2 Correct Answers

One set of results (Could be very different each run!)

[('101', 277), ('110', 258), ('001', 252), ('010', 237)]   ** Possible Error in Example Code, expected two values only with equal probability **

1024 total runs on qiskit simulator

277 results in state: $|101\rangle$

258 Results in state: $|110\rangle$

** On a real quantum hardware, you will get all 8 possibilities and their counts!

Part 5: Questions / Comments / Complaints

Special Thanks

Dr. Samuel Lomanoco

Dr. Charles Nicholas

Dr. Edward Raff

Ajinkya Borle

Dan Strano

Ramin Ayanzadeh

Thanks!

Contact: allgood1@umbc.edu , nick.allgood@gmail.com

Github: https://github.com/nallg00d

Simon's Research

Master's Research

References

[1] IBM. *IBM Quantum Experience.* https://quantum-computing.ibm.com

[2] Rieffel, Eleanor Polak, Wolfgang. *Quantum Computing A Gentle Introduction.* The MIT Press Cambridge, MA. 2011 Massachusetts Institute of Technology. ISBN 978-0-262-01506-6.

[3] Nielsen, Michael A. Chuang, Isaac L. *Quantum Computation and Quantum Information.* 2010 Cambridge University Press, New York. ISBN 978-1-107-00217-3

[4] Matzke, Douglas J. *Quantum Computing Using Geometric Algebra*. Published 2002. Accessed Aug. 5, 2019. http://www.matzkefamily.net/doug/papers/PHD/phd_matzke_final.pdf

[5] Desjardins, Jeff. *The Three Types of Quantum Computers and Their Uses*. Visual Capitalist. Published March 14, 2016. Accessed August 7, 2019. https://www.visualcapitalist.com/three-types-quantum-computers/

References (cont.)

[6] D-Wave Inc. *D-Wave Announces Quantum Hybrid Strategy and General Availability of D-Wave Hybrid.* Published June 26, 2019. Accessed August 7, 2019. https://www.dwavesys.com/press-releases/d-wave-announces-quantum-hybrid-strategy-and-general-availability-d-wave-hybrid

[7] Strano, Daniel. Bollay, Benn. V*M6502q and Qrack*. https://github.com/vm6502q/qrack https://qrack.readthedocs.io/en/latest/

[8] Microsoft Research. *The Q# Programming Language.* https://docs.microsoft.com/en-us/quantum/language/?view=qsharp-preview

[9] IBM Research Quiskit Community. *Qisket*. https://qiskit.org/

[10] D-Wave Inc. *Quantum Annealing Applied to Optimization Problems in Radiation Medicine.* https://www.dwavesys.com/sites/default/files/Radiotheraphy-Optimization-Roswell-Park_0.pdf

[11] D-Wave Inc. Booz-Allen Hamilton. *Heterogeneous Quantum Computing For Satellite Communication.* https://www.dwavesys.com/sites/default/files/QuantumForSatellitesQubits-4.pdf

[12] D-Wave Inc. Volkswagen. *Quantum Computing at Volkswagen: Traffic Flow Optimization using the D-Wave Quantum Annealer.* https://www.dwavesys.com/sites/default/files/VW.pdf

[13] D-Wave Inc. Accenture. *Quantum Computing Advance Drug Discovery. Accenture.* https://www.accenture.com/us-en/success-biogen-quantum-computing-advance-drug-discovery

[14] Josza, Richard. *An Introduction to Measurement Based Quantum Computing.* Department of Computer Science. Bristol, BS8 1UB UK. https://arXiv:quant-ph/0508124v2 . Published Sept 20, 2005. Accessed August 13, 2019.

[15] IBM. Qiskit. https://qiskit.org/