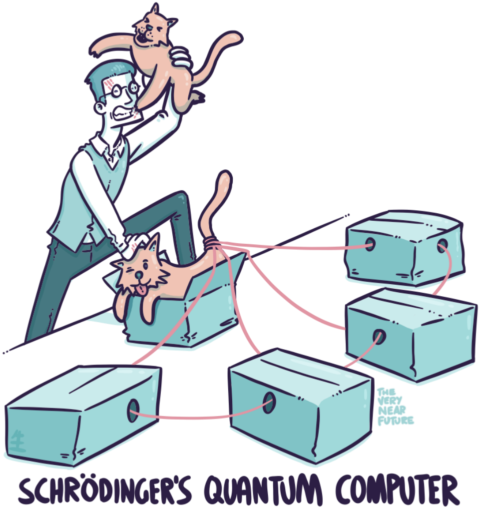
Intro to Quantum Algorithms

Mark Boady

Department of Computer Science
Drexel University

May 5, 2021

Quantum Computers



https://www.reddit.com/r/QuantumComputing/comments/glenar/comic_dead_or_9_lives/

Quantum Computers in the News

- Is Quantum Computing Placing Bitcoin's Future in Jeopardy? (May 1, 2021 The Daily Hodl)
- A student's physics project could make quantum computers twice as reliable (April 12, 2021 Live Science)
- Cryptographers are Racing Against Quantum Computers (April 20, 2021 builtin)
- IBM promises 1000-qubit quantum computer - a milestone - by 2023 (Sept 15, 2020 Science Mag)

- The future is bright
- There are a lot of hardware limitations right now
- In the News: Factoring Large integers will destroy cryptography as we know it
- In Reality: Python can probably still do better right now

Shor's Algorithm

- The most famous quantum algorithm.
- Can factor integers!
- Largest Number Ever Factored: 21 in 2012

  Martin-Lopez, Enrique; MartÃn-Lopez, Enrique; Laing, Anthony; Lawson, Thomas; Alvarez, Roberto; Zhou, Xiao-Qi; O'Brien, Jeremy L. (12 October 2012). "Experimental realization of Shor's quantum factoring algorithm using qubit recycling". Nature Photonics. 6 (11): 773-776.
- A failed attempt was made to factor 35 in 2019

  Amico, Mirko; Saleem, Zain H.; Kumph, Muir (2019-07-08). "An Experimental Study of Shor's Factoring Algorithm on IBM Q". Physical Review A. 100 (1): 012305.

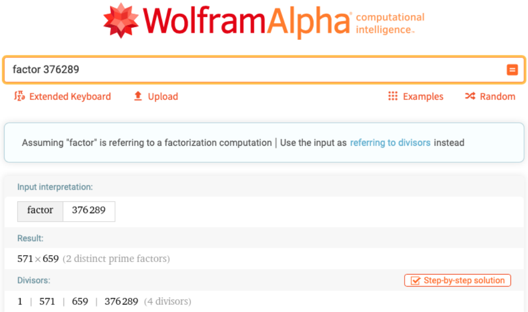
Quantum Annealing

- Uses special hardware (not universal computation)
- Shown to factor: 15, 143, 59989, and 376289

  Jiang, S., Britt, K.A., McCaskey, A.J. et al. Quantum Annealing for Prime Factorization. Sci Rep 8, 17667 (2018). https://doi.org/10.1038/s41598-018-36058-z
- Hardware designed for optimization problems limits uses

Classical Computers

- It probably isn't worth your effort to factor 21, 35, or even 376289 on a quantum computer.



Where we stand now

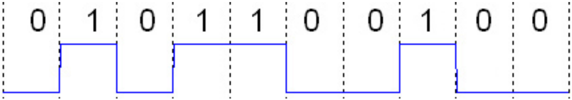
- We can analyze and test algorithm at small scales.
- We know they work.
- We know they have amazing potential.
- Significant Hardware and Software limitations right now.
- We are in the 1950s of Quantum Computing.
- Learn the basics now to be prepared for the near future.
- We are advancing much faster this time around!

Today's Goals

- What is a Quantum Computer?
- How is it different from a Classical Computer?
- How do we to write simple programs?
- How does the Deutsch-Jozsa Algorithm work?

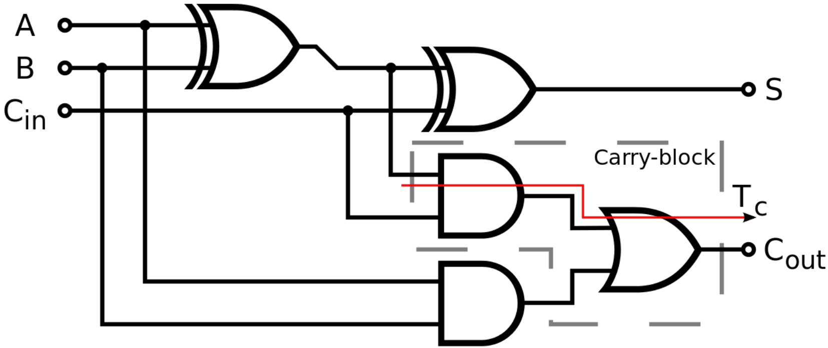
Bits

- Classical Computing is built on the Bit.
- 0 (Low Voltage 0-2 Volts)
- 1 (High Voltage 3-5 Volts)
- Bits and Circuits are governed by Classical Physics.
- These values are **discrete**.



El pak at English Wikipedia `https://commons.wikimedia.org/wiki/File:Original_message.jpg`

- We apply logic gates to bits to create circuits.
- A logic gate takes input bits and produces and output bit.



https://commons.wikimedia.org/wiki/File:Full-adder_logic_diagram.svg

Quantum Computing

- The **Qubit** is the quantum equivalent of a bit
- Bit becomes **Qubit**
- Logic becomes **Linear Algebra**
- Classical Physics becomes **Quantum Physics**
- Writing Quantum Algorithms requires a completely different perspective than Classical Algorithms

Qubit

- When **measured** a qubit can have one of two values
  - $|0\rangle$ the quantum **False** or 0 state
  - $|1\rangle$ the quantum **True** or 1 state
- This is called *Dirac Notation*.
- During computation a qubit can be in **superposition**

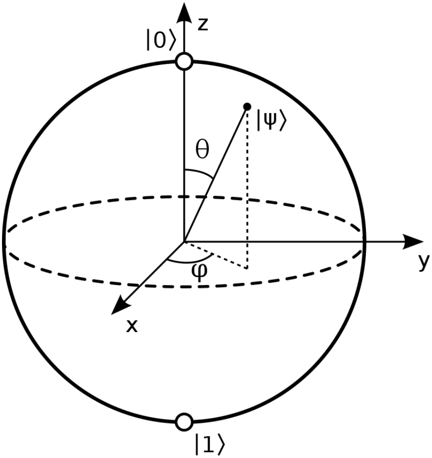
$$|\psi\rangle = \alpha |0\rangle + \beta |1\rangle$$

- A **superposition** is a linear combination of states.

Bloch Sphere

- Imagine a Qubit is a ball.
- We can point to any position on the ball.
- When we **measure** the location we get either 0 or 1.
- The top of the sphere always **measures** 0.
- The bottom of the sphere always **measures** 1.
- Every other location has some probability of 0 and some probability of 1.

Bloch Sphere



Smite-Meister `https://commons.wikimedia.org/wiki/File:Bloch_sphere.svg`

Qubit

- A qubit is a linear combination of states.

$$|\psi\rangle = \alpha |0\rangle + \beta |1\rangle$$

- α and β are **complex numbers**
- When the qubit is measured it will be either 0 or 1
- We can guess the probabilities by repeated experiments
- α and β determine the exact point on the sphere
- The probability of measuring 0 is $|\alpha|^2$
- The probability of measuring 1 is $|\beta|^2$
- Since both are probabilities then

$$|\alpha|^2 + |\beta|^2 = 1$$

Superposition

- A qubit has a *continuum* of states be $|0\rangle$ and $|1\rangle$
- We can pick out point on the sphere at a position that is sometimes in the 1 side and sometimes in the 0 side.
- When **measured** the qubit will either be in state 0 or 1
- A qubit with a 50/50 split between 0 and 1

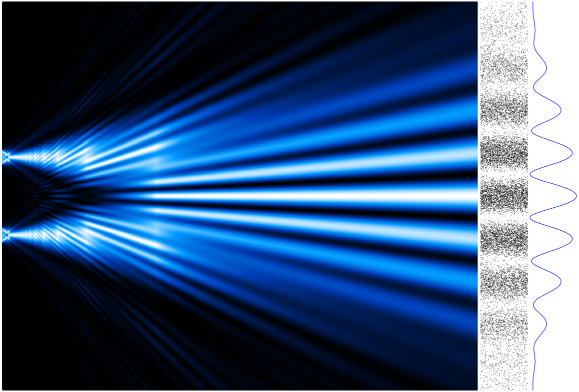
$$|+\rangle = \frac{1}{\sqrt{2}} |0\rangle + \frac{1}{\sqrt{2}} |1\rangle$$

$$|\alpha|^2 = \left|\frac{1}{\sqrt{2}}\right|^2 = \frac{1}{2} = 50\%$$

$$|\beta|^2 = \left|\frac{1}{\sqrt{2}}\right|^2 = \frac{1}{2} = 50\%$$

Interference

- Interference - probability *waves* can cancel out.



By Alexandre Gondran - Own work, CC BY-SA 4.0,

https://commons.wikimedia.org/w/index.php?curid=53628849

Vector Representation

- A quantum circuit is a vector

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle = \begin{bmatrix} \alpha \\ \beta \end{bmatrix}$$

- A multiple qubit circuit is a larger vector.
- A two-qubit system has a probability for each result

$$\begin{aligned} |\psi\rangle =& \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle \\ =& \begin{bmatrix} \alpha_{00} \\ \alpha_{01} \\ \alpha_{10} \\ \alpha_{11} \end{bmatrix} \end{aligned}$$

- Qiskit is IBM's Python 3 Library for programming quantum computers
- Allows for both simulation and execution on real hardware
- First Install Anaconda
  `https://www.anaconda.com`
- Install Qiskit
  `https://qiskit.org/documentation/getting_started.html`

Libraries

- We need to import the correct libraries
- qiskit has the all the basics to make circuits
- BasicAer has a simulator
- You can simulator or connect to a real QC

code/circuit1.py

```
#Mark Boady - 2021
#Intro to Quantum Computers

#Import the Libraries
from qiskit import *
#For simulations:
from qiskit import BasicAer
```

- We need to make a circuit.
- First number is how many Qubits (3)
- Second number is how many Classic Bits (3)
- We need to **measure** into Classic Bits to see the results.

code/circuit1.py

```
#I want a circuit with
#3 Qubits and 3 classic bits
qc = QuantumCircuit(3,3)
```

Add X Gate

- We add an X gate.
- Similar to **not**
- Flips the values of α and β
- Flips the probabilities of 1 and 0 as outcomes

code/circuit1.py

```
#Add a gate
qc.x(0)
```

Measure the Results

- We need to measure the results.
- Measure a Qubit into a Classic bit.
- The **barrier** is just for visuals, it makes the diagram easier to read.

code/circuit1.py

```
#We need to measure to see the results
#Barrier is just for visual
qc.barrier(range(0,3))
qc.measure(0,0)
qc.measure(1,1)
qc.measure(2,2)
```

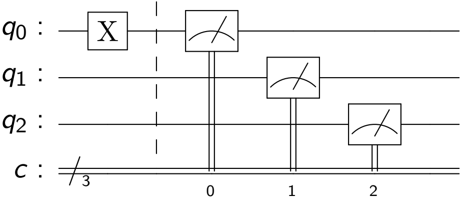
Draw the Circuit

- We can ask qiskit to draw the circuit

code/circuit1.py

```
#Print as text
print(qc.draw(output="text"))
#Latex for Slides
print(qc.draw(output="latex_source"))
#Matplot to make an image
qc.draw(output="mpl",filename="cicuit.png")
```

Circuit with X Gate



Simulate

- We simulate the circuit
- Since results can be random, we run many tests.

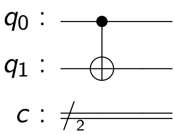
code/circuit1.py

```
#Run our simulation!
#Create Simulator
backend_sim = BasicAer.get_backend('
    qasm_simulator')
#Run 2,048 tests
job_sim = execute(qc, backend_sim, shots=2048)
#Get the results
result_sim = job_sim.result()
#Show the count of each outcome
counts = result_sim.get_counts()
print(counts)
```

Results

- The results are: {'001': 2048}
- q_0 was a 1 on every single test measurement
- q_1 was a 0 on every single test measurement
- q_2 was a 0 on every single test measurement

- We need multiple bit gates to do anything useful.
- The CNOT gate is a conditional gate on two bits.
- If *control qubit* is one then apply X to *target qubit*
- If q_0 then apply X to q_1



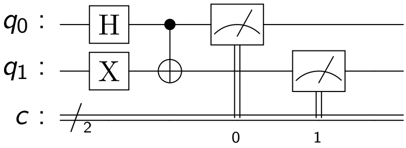
- Another way to example this gate uses the XOR
- Exclusive Or: True when two inputs are different and False otherwise.
- $q_0 = q_0$
- $q_1 = q_0 \oplus q_1$

code/cnot.py

```
#CNOT Example
qc = QuantumCircuit(2,2)
#Add a gate
qc.cnot(0,1)
```

The Hadamard Gate

- The Hadamard Gate puts a Qubit into superposition
- It has a 50% chance of being 1 and a 50% chance of being 0



- We put q_0 into 50/50 superposition
- We start q_1 as 1
- We then apply the CNOT

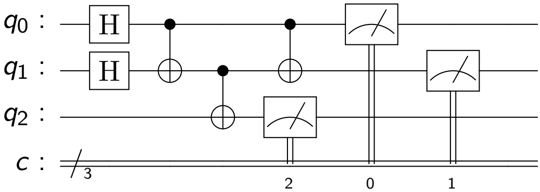
code/hcnot.py

```
#CNOT Example
qc = QuantumCircuit(2,2)
#Add a gate
qc.h(0)#Superposition q0
qc.x(1)#Start q1 as 1
qc.cnot(0,1)
#Measure Results
qc.measure(0,0)
qc.measure(1,1)
```

Results

- Results: {'10': 1002, '01': 1046}
- 1002 tests caused
  - q_0 was 0
  - q_1 stayed 1
- 1046 tests caused
  - q_0 flipped to 1
  - q_1 was flipped to 0 by the CNOT

- Applying CNOT twice undoes the operation.
- q_0 and q_1 stay the same.
- q_3 is the result of $q_0 \oplus q_1$



Results

- We can imagine our results are a Truth Table
- We made the Truth Table for XOR

q_0	q_1	q_2	Count
0	0	0	525
1	0	1	516
0	1	1	480
1	1	0	527

- How can we use this?
- We need to create interference
- We can only measure classic bits
- Interference can effect the probabilities
- We need to make interference cancel out wrong answers
- We will be left with the right answer

Problem Statement

- You are given a Boolean Function with n variables
- You are promised one of the following:
  - The function is always True (Tautology)
  - The function is always False (Contradictory)
  - The function is split 50% False and 50% True
- Problem: Which of the three options is the function?

Classical Solution

- An n variable truth table has 2^n rows
- If we test 1 more than half we will know the answer.
  1. All true must be Tautology
  2. All false must be Contradictory
  3. We got at 2 different answers, it must be 50/50 split
- We **must** do $\frac{1}{2} * (2^n) + 1 = 2^{n-1} + 1$ tests.
- You cannot do better on a classical computer.

Deutsch-Jozsa Algorithm

- We can do better on a quantum computer
- We make a circuit with $n+1$ qubits.
- We put the Boolean function on the first n qubits and the answer on the last qubit.
- XOR is a boolean function, we can use the example from earlier!
- We create interference on the qubits

Deutsch-Jozsa Algorithm

- Step 1: Apply Hadamard's gate to qubits representing the function input

code/deutsch_xor.py

```
#3-qubits and 2 classic
qc = QuantumCircuit(3,2)

#Build a Circuit
#Step 1: Apply H to the input bits
qc.h(0)
qc.h(1)
```

- Step 2: Apply X then Hadamard's gate to the qubit representing the result

code/deutsch_xor.py

```
#Step 2: Apply X to the result bit
#then apply H
qc.x(2)
qc.h(2)
```

- Step 3: Implement the Boolean function

code/deutsch_xor.py

```
#Step 3: Implement Boolean Function
qc.barrier()
qc.cnot(0,1)
qc.cnot(1,2)
qc.cnot(0,1)
qc.barrier()
```

- Step 4: Apply H to input qubits (undo original H)

code/deutsch_xor.py

```
#Step 4: Apply H to all the inputs
qc.h(0)
qc.h(1)
```

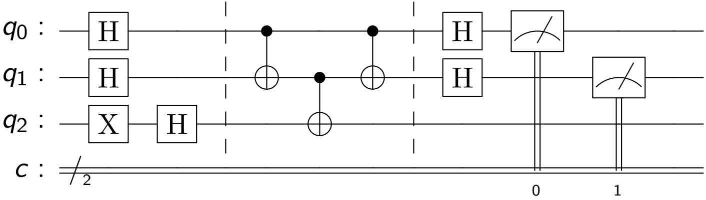
Deutsch-Jozsa Algorithm

- Step 5: Measure the input bits

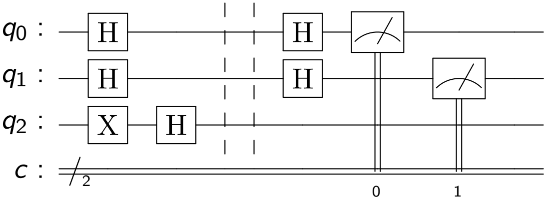
code/deutsch_xor.py

```
#Step 5: Measure inputs
qc.measure(0,0)
qc.measure(1,1)
```

- Results: {'11': 2048}



- What if we take away the Boolean function?
- Results: {'00': 2048}



Results

- If the Boolean function is a 50/50 split then the input bits **will** all measure 1
- If the Boolean function is constant then the input bits **will** all measure 0
- Interference cancels out all other outcomes
- Once we know a function is constant, determining if it is a Tautology or Contradictory is trivial

Why

- What? Why? I'm confused?
- To see why the probabilities cancel out, we would need to examine the linear algebra
- Above the scope of this talk.

Famous Algorithms

- Grover's algorithm
  - Given a Boolean function, find a way to make it true
  - Classic Solution is $O(2^n)$
  - Grover is $O(\sqrt{2^n})$
- Shor's Algorithm
  - Given an integer, find a prime factor
  - Classical Solution is $O\left(\mathrm{e}^{1.9(\log n)^{1/3}(\log\log N)^{1/3}}\right)$
  - Shor is $O\left((\log N)^2(\log\log N)(\log\log\log N)\right)$

Where Do We Go From Here

- Mark Boady mwb33@drexel.edu
- Qiskit Tutorial: https://qiskit.org/documentation/tutorials/circuits/1_getting_started_with_qiskit.html
- Qiskit Textbook: https://qiskit.org/textbook/preface.html
- Coding With Qiskit: https://www.youtube.com/playlist?list=PLOFEBzvs-Vvp2xg9-POLJhQwtVktlYGbY
- Quantum Computation and Quantum Information by Michael A. Nielsen and Isaac L. Chaung