Bernstein-Varirani Algorithm and Quantum Fourier Transforms

Julie Butler

Algorithm 1: Bernstein-Varirani Algorithm

Problem Statement and Classical Implementations

Computational Performance and Efficiency

- Computational performance is a measure of how the time needed to run a program changes as you increase the size of the problem.
- **Example:** Polynomial time scaling $t = O(n^k)$ for a set k
- **Example:** Exponential time scaling $t = O(e^n)$
- We say and algorithm is **efficient** if the number of steps it needs to solve a problem scales polynomially with respect to the size of the problem.

Query Method

- For some methods it does not make sense to discuss the amount of time the algorithm took but instead to talk about performance in terms of the number of queries the algorithm made to a given quantity
  - How many times was a set quantity referenced and used in the code to solve the problem

A Problem Statement

Consider a function, f, which has the following form:

$$f(x) = x \cdot s \ mod\ 2$$

which will map a string of n binary digits (x) to a single binary digit (f(x)). The variable s is a "secret string" of n binary digits

Modulo Review:

$$0 \ mod\ 2 = 0 \quad 1 \ mod\ 2 = 1$$

Modulo in Python is %

Forward Direction

Example: Let $x_1 = 100$, $x_2 = 001$, and $x_3 = 111$. Let the secret string, s, be $s = 101$. Compute f(x) for x_1, x_2, and x_3.

Reverse Direction??

Example: Consider the following inputs and outputs for f(x). What is s?

- $f(00) = 0$
- $f(01) = 0$
- $f(10) = 1$
- $f(11) = 1$
- Not really as easy as the initial problem.

Absolute Best Case Classical Implementation

Let's consider the test case where the length of s is 3, $s = s_1 s_2 s_3$.

Now consider the vectors $x_1 = 100$, $x_2 = 010$, and $x_3 = 001$.

$$x_1 \cdot s = s_1 \quad x_2 \cdot s = s_2 \quad x_3 \cdot s = s_3$$

Generalization: For a secret string of length n, you will need n vectors to determine its components $\longrightarrow$ you need to make n queries to s.
AND THIS IS BEST CASE!!!

Worst Case: Try values of s until your find the correct one (brute force method), possibly infinite queries to s

Python Implementation

See the associated Jupyter notebook for these slides for the Python implementation.

Quantum Implementation

A Greater Look at the Two Computational Bases

- Our First Computational Basis

$$|\uparrow\rangle = |0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad |\downarrow\rangle = |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

- Our Second Computational Basis

$$|+\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \quad |-\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

How are they related?

$$|+\rangle = \frac{1}{\sqrt{2}}(|\uparrow\rangle + |\downarrow\rangle)$$
$$|-\rangle = \frac{1}{\sqrt{2}}(|\uparrow\rangle - |\downarrow\rangle)$$

How do they interact with Hadamard Gates?

- From our slides last week

$$H|\uparrow\rangle = |+\rangle$$

$$H|\downarrow\rangle = |-\rangle$$

- But what about the other basis?
  - To the board!!

Summary of Important Equations

$$|\uparrow\rangle = |0\rangle = \begin{bmatrix}1\\0\end{bmatrix} \quad |\downarrow\rangle = |1\rangle = \begin{bmatrix}0\\1\end{bmatrix}$$

$$|+\rangle = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\1\end{bmatrix} \quad |-\rangle = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\-1\end{bmatrix}$$

$$|+\rangle = \frac{1}{\sqrt{2}}(|\uparrow\rangle + |\downarrow\rangle)$$

$$|-\rangle = \frac{1}{\sqrt{2}}(|\uparrow\rangle - |\downarrow\rangle)$$

$$H|\uparrow\rangle = |+\rangle \quad H|\downarrow\rangle = |-\rangle$$

$$H|+\rangle = |\uparrow\rangle \quad H|-\rangle = |\downarrow\rangle$$

Double Hadamard Gate

- What happens if we apply a Hadamard gate twice to $|\uparrow\rangle$ or $|\downarrow\rangle$?

$$H(H|\uparrow\rangle) = H|+\rangle = |\uparrow\rangle$$

$$H(H|\downarrow\rangle) = H|-\rangle = |\downarrow\rangle$$

- Applying the Hadamard gate twice gives you back the original state

A Deeper Exploration to NOT and CNOT Gates

$$X|\uparrow\rangle = |\downarrow\rangle \quad X|\downarrow\rangle = |\uparrow\rangle$$

$$X|+\rangle = ???$$

A Deeper Exploration to NOT and CNOT Gates

$$X|\uparrow\rangle = |\downarrow\rangle \quad X|\downarrow\rangle = |\uparrow\rangle$$

$$X|+\rangle = \frac{1}{\sqrt{2}}(X|\uparrow\rangle + X|\downarrow\uparrow)$$

$$= \frac{1}{\sqrt{2}}(|\downarrow\rangle + \uparrow\rangle) = \frac{1}{\sqrt{2}}(|\uparrow\rangle + |\downarrow\rangle)$$

$$= |+\rangle$$

A Deeper Exploration to NOT and CNOT Gates

$$X|\uparrow\rangle = |\downarrow\rangle \quad X|\downarrow\rangle = |\uparrow\rangle$$

$$X|+\rangle = |+\rangle$$

$$X|-\rangle = ???$$

A Deeper Exploration to NOT and CNOT Gates

$$X|\uparrow\rangle = |\downarrow\rangle \quad X|\downarrow\rangle = |\uparrow\rangle$$

$$X|+\rangle = |+\rangle$$

$$X|-\rangle = \frac{1}{\sqrt{2}}(X|\uparrow\rangle - X|\downarrow\rangle) = \frac{1}{\sqrt{2}}(|\downarrow\rangle - |\uparrow\rangle)$$

$$= \frac{1}{\sqrt{2}}(-|\uparrow\rangle + |\downarrow\rangle) = \frac{-1}{\sqrt{2}}(|\uparrow\rangle - |\downarrow\rangle)$$

$$= -|-\rangle$$

A Deeper Exploration to NOT and CNOT Gates

$$X|\uparrow\rangle = |\downarrow\rangle \quad X|\downarrow\rangle = |\uparrow\rangle$$

$$X|+\rangle = |+\rangle \quad X|-\rangle = -|-\rangle$$

Do not lose the negative sign, it will be important soon!

A Deeper Exploration to NOT and CNOT Gates

- Now time to explore the CNOT gates on different combinations of bits
- Notation:

$$C_X|target\ qubit \quad control\ qubit\rangle$$

- Different order than how you input it into Qiskit, this is the order IBM uses for their documentation (even though they wrote Qiskit...)
- Different sources use different notations/orders so just be careful and make sure you always know what you are looking at

A Deeper Exploration to NOT and CNOT Gates

$$C_X|\uparrow\uparrow\rangle = |\uparrow\uparrow\rangle$$
$$C_X|\uparrow\downarrow\rangle = |\downarrow\downarrow\rangle$$
$$C_X|\downarrow\downarrow\rangle = |\uparrow\downarrow\rangle$$
$$C_X|\downarrow\uparrow\rangle = |\downarrow\uparrow\rangle$$

- CNOT gates create entanglements
- Note that in the above equations, only the target qubit changed states.

A Deeper Exploration to NOT and CNOT Gates

- Quick aside: entangled states
- We want to write all entangled states in terms of the $|\uparrow\rangle$ and $|\downarrow\rangle$ basis
- The four entangled states that already exist in that basis are: $|\uparrow\uparrow\rangle$, $|\uparrow\downarrow\rangle$, $|\downarrow\uparrow\rangle$, and $|\downarrow\downarrow\rangle$

A Deeper Exploration to NOT and CNOT Gates

- What about entanglements that mix with the other basis?

$$|\uparrow +\rangle = \frac{1}{\sqrt{2}}(|\uparrow\uparrow\rangle + |\uparrow\downarrow\rangle)$$

$$|\uparrow -\rangle = \frac{1}{\sqrt{2}}(|\uparrow\uparrow\rangle - |\uparrow\downarrow\rangle)$$

$$|+\uparrow\rangle = \frac{1}{\sqrt{2}}(|\uparrow\uparrow\rangle + |\downarrow\uparrow\rangle)$$

$$|-\uparrow\rangle = \frac{1}{\sqrt{2}}(|\uparrow\uparrow\rangle - |\downarrow\uparrow\rangle)$$

A Deeper Exploration to NOT and CNOT Gates

- Entanglement Continues

$$|\downarrow +\rangle = \frac{1}{\sqrt{2}}(|\downarrow\uparrow\rangle + |\downarrow\downarrow\rangle)$$

$$|\downarrow -\rangle = \frac{1}{\sqrt{2}}(|\downarrow\uparrow\rangle - |\downarrow\downarrow\rangle)$$

$$|+\downarrow\rangle = \frac{1}{\sqrt{2}}(|\uparrow\downarrow\rangle + |\downarrow\downarrow\rangle)$$

$$|-\downarrow\rangle = \frac{1}{\sqrt{2}}(|\uparrow\downarrow\rangle - |\downarrow\downarrow\rangle)$$

A Deeper Exploration to NOT and CNOT Gates

- What about entanglements only in the superposition basis?

$$|++\rangle = \frac{1}{\sqrt{2}}(|+\uparrow\rangle + |+\downarrow\rangle)$$

$$= \frac{1}{\sqrt{2}}(\frac{1}{\sqrt{2}}(|\uparrow\uparrow\rangle + |\downarrow\uparrow\rangle) + \frac{1}{\sqrt{2}}(|\uparrow\downarrow\rangle + |\downarrow\downarrow\rangle))$$

$$= \frac{1}{2}(|\uparrow\uparrow\rangle + |\uparrow\downarrow\rangle + |\downarrow\uparrow\rangle + |\downarrow\downarrow\rangle)$$

A Deeper Exploration to NOT and CNOT Gates

- More Entanglement

$$|--\rangle = \frac{1}{\sqrt{2}}(|-\uparrow\rangle - |-\downarrow\rangle)$$

$$= \frac{1}{\sqrt{2}}(\frac{1}{\sqrt{2}}(|\uparrow\uparrow\rangle - |\downarrow\uparrow\rangle) - \frac{1}{\sqrt{2}}(|\uparrow\downarrow = |\downarrow\downarrow\rangle))$$

$$= \frac{1}{2}(|\uparrow\uparrow\rangle - |\uparrow\downarrow\rangle - |\downarrow\uparrow\rangle + |\downarrow\downarrow\rangle)$$

A Deeper Exploration to NOT and CNOT Gates

- More Entanglement

$$|+-\rangle = \frac{1}{\sqrt{2}}(|+\uparrow\rangle - |+\downarrow\rangle)$$

$$= \frac{1}{\sqrt{2}}(\frac{1}{\sqrt{2}}(|\uparrow\uparrow\rangle + |\downarrow\uparrow\rangle) - \frac{1}{\sqrt{2}}(|\uparrow\downarrow\rangle + |\downarrow\downarrow\rangle))$$

$$= \frac{1}{2}(|\uparrow\uparrow\rangle - |\uparrow\downarrow\rangle + |\downarrow\uparrow\rangle - |\downarrow\downarrow\rangle)$$

A Deeper Exploration to NOT and CNOT Gates

- One final slide of entanglement

$$|-+\rangle = \frac{1}{\sqrt{2}}(|-\uparrow\rangle + |-\downarrow\rangle)$$

$$= \frac{1}{\sqrt{2}}(\frac{1}{\sqrt{2}}(|\uparrow\uparrow\rangle - |\downarrow\uparrow\rangle) + \frac{1}{\sqrt{2}}(|\uparrow\downarrow\rangle - |\downarrow\downarrow\rangle))$$

$$= \frac{1}{2}(|\uparrow\uparrow\rangle + |\uparrow\downarrow\rangle - |\downarrow\uparrow\rangle - |\downarrow\downarrow\rangle)$$

A Deeper Exploration to NOT and CNOT Gates

- What if the target qubit is a superposition?

$$C_X|+\downarrow\rangle = \frac{1}{\sqrt{2}}(C_X|\uparrow\downarrow\rangle + C_X|\downarrow\downarrow\rangle)$$

$$= \frac{1}{\sqrt{2}}(|\downarrow\downarrow\rangle + |\uparrow\downarrow\rangle)$$

$$= |+\downarrow\rangle$$

- In this example there was no change, but try other combinations!

A Deeper Exploration to NOT and CNOT Gates

- What if the control qubit is a superposition?

$$C_X|\uparrow +\rangle = \frac{1}{\sqrt{2}}(C_X|\uparrow\uparrow\rangle + C_X|\uparrow\downarrow\rangle)$$

$$= \frac{1}{\sqrt{2}}(|\uparrow\uparrow\rangle + |\downarrow\downarrow\rangle)$$

$$= |\Phi^+\rangle$$

- We made a Bell State!

A Deeper Exploration to NOT and CNOT Gates

- What if the target and control qubits are superpositions?

$$C_X|-+\rangle = \frac{1}{2}(C_X|\uparrow\uparrow\rangle + C_X|\uparrow\downarrow\rangle - C_X|\downarrow\uparrow\rangle - C_X|\downarrow\downarrow\rangle)$$

$$= \frac{1}{2}(|\uparrow\uparrow\rangle + |\downarrow\downarrow\rangle - |\downarrow\uparrow\rangle - |\uparrow\downarrow\rangle)$$

$$= \frac{1}{2}(|\uparrow\uparrow\rangle - |\uparrow\downarrow\rangle - |\downarrow\uparrow\rangle + |\downarrow\downarrow\rangle)$$

$$|--\rangle$$

- ...we changed the control qubit, not the target qubit...

Phase Kickback

- Applying the CNOT gate to certain combinations of entangled states will change the **control qubit** and not the **target qubit**
  - The target qubit kicks the *phase* up to the control qubit
  - This is called *phase kickback*
- This will be a feature of almost every quantum algorithm in this course, including the Bernstein-Varirani Algorithm

What is an Oracle?

- An oracle is a black box that takes information and somehow produces a result
  - Bernstein-Varirani is called a black box algorithm
- **But there will be no black boxes in this class!**

How Could We Go About Making A Quantum Algorithm for the Above Problem Statement?

- Likely want to start by creating some superpositions (so Hadamard gates)
- Would make sense to have one qubit per digit we are trying to find
- Qubits will go into an oracle/black box and come out representing the hidden string, one more set of Hadamard gates to convert back to 0's and 1's
- What Could the black box be?
  - ...something involving phase kickback?

To the board!!

Drawbacks?

- All of the steps with the qubits are happening simultaneously (quantum parallelism!)
- Does have linear speedup, only one query of the secret string needed
- Need one qubit per digit in the secret string plus one extra qubit it act as the auxiliary qubit
  - Not super efficient in terms of scaling
- Daniel Simon wanted to prove this was inefficient and accidentally created an algorithm which is **exponentially faster** on a quantum computer
  - This is the basis of Shor's algorithm we will cover in a few weeks

Qiskit Implementation

See the Jupyter notebook associated with the slides!

Conclusions

- Is the algorithm useful?
  - No! It is essentially just a toy/demonstration model
- Did we learn something important?
  - Yes! Phase kickback will be used through out many of the algorithms used in this class and the study of this algorithm led to the discover of many others.

Quantum Fourier Transforms

Euler's Identity

$$e^{i\theta} = cos\theta + isin\theta$$

$$cos^2\theta + sin^2\theta = 1$$

Discrete Fourier Transform (DFT) (Words Explanation)

- Applies a Fourier transform to discrete data $\longrightarrow$ represent complicated data in terms of a sum of trig functions
  - Discrete data here meaning only existing at certain points
- Essentially the DFT takes a set of data and transforms it into a new set of data by multiplying the data by a series of trigonometric coefficients
  - Applications in a few slides!

Discrete Fourier Transform (Math)

- Given a set of discrete, complex numbers, x, these can be mapped to another set of discrete, complex numbers, X using:

$$X_k = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} x_n W_{kn}$$

- W_{kn} is an element on the DFT matrix (next slide) which is composed on imaginary exponential terms (trig functions)
- Be aware that there is also an inverse DFT algorithm which takes X and creates x.
  - Sometimes the above equation is referred to as the forward transform, sometimes the reverse...

Discrete Fourier Transform Matrix

$$W = \frac{1}{\sqrt{N}} \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \omega^3 & \cdots & \omega^{N-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \cdots & \omega^{2(N-1)} \\ 1 & \omega^3 & \omega^6 & \omega^9 & \cdots & \omega^{3(N-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{N-1} & \omega^{2(N-1)} & \omega^{3(N-1)} & \cdots & \omega^{(N-1)(N-1)} \end{bmatrix},$$

Figure 1: DFT Matrix

- In the above matrix $\omega = e^{-2\pi i/N}$, where N is the size of the matrix
- Note that some resources do not include the normalization factor

Discrete Fourier Transform Matrix

- **Exercise 1:** Derive the DFT matrix for $N = 2$. Does it look familiar?
- **Exercise 2:** Derive the DFT matrix for $N = 3$.
- **General Rule:** For any given value of N, W is unitary.
  - Why could this be important?

Applications of Discrete Fourier Transform

- Many applications across all fields of science, but applications for the every day person include:
  - Data compression
  - For sound data, returns the frequencies of the sound
    - Gives a spike for every note, can then remove unwanted notes
  - Similar idea can be applied to remove noise from images
- **General Rule:** Does is oscillate? If yes then you can Fourier it!

Classical Implementation

See Jupyter Notebook associated with these slides!!

Fast Fourier Transform (FFT)

- Discrete Fourier Transform: $O(N^2)$
  - N outputs where each outputs is a sum of N terms
- Fast Fourier Transform (Best): $O(NlogN)$
  - Use techniques such as recursion to reduce using a sum of N terms for each transformation
  - **Do not need to know how this works!** Just be aware it exists.

Quantum Implementation

Road Map

1. Theory: What is the quantum Fourier transform (QFT) and what does it do?
2. Applications: Why do we want to perform a fourier transform on a quantum system?
3. Constructing a circuit: What gates could be used to create a Fourier transform on a quantum circuit?
4. Qiskit implementation

Theory

- Same idea: we want to take a set of data and transform it by multiplying it with different scale factors
- Consider a quantum system made up of basis states $|k\rangle$
  - For example, for the two level system $\{|k\rangle\} = |\uparrow\rangle, |\downarrow\rangle$
- Any quantum state can then be written as, where α_k is a (possibly) complex constant:

$$|\psi\rangle = \sum_k \alpha_k |k\rangle$$

Theory Continued

- Then performing a Quantum Fourier Transform (QFT) on the state $|\psi\rangle$ results in a new state:

$$|\psi\rangle_{QFT} = \sum_k \beta_k |k\rangle$$

- The coefficients β_k can be defined a follows, where α_k are the coefficients of the original state

$$\beta_k = \frac{1}{\sqrt{2^n}} \sum_j W_{jk} \alpha_k$$

- In the above definition $N = 2^n$

Applications

- Most algorithms we will look at in this class involve a QFT at some point
- Why?
  - QFT finds periodicity (patterns)
    - Think trig functions

Constructing a Circuit: New Gates

- Phase Gates:

$$S = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$$

$$P(\phi) = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\phi} \end{bmatrix}$$

Constructing a Circuit: New Gates

- Rotation Gate:

$$R_x(\theta) = \begin{bmatrix} cos(\theta/2) & -isin(\theta/2) \\ -isin(\theta/2) & cos(\theta/2) \end{bmatrix}$$

$$R_y(\theta) = \begin{bmatrix} cos(\theta/2) & -sin(\theta/2) \\ sin(\theta/2) & cos(\theta/2) \end{bmatrix}$$

$$R_z(\theta) = \begin{bmatrix} e^{i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{bmatrix}$$

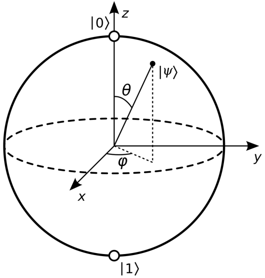
New Gates Part 2

- Controlled Phase Gate:

$$CPhase(\phi) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\phi} \end{bmatrix}$$

Bloch Sphere

- Reminder about the Bloch sphere visualization



New Gates and Functions in Qiskit

- New gates and new Qiskit functionality
  - To Jupyter!

Encoding Binary Numbers into a Quantum Computer

- Binary digits are either 0 or 1, qubits can either be 0 (up) or 1 (down)
- 'bin' is a built in function in Python to convert a digit to its binary form
- The goal of QFT is to represent a number in binary using states other than up/down (different phases)
- **To Jupyter!**

Note on Notation

- We say that performing a quantum Fourier transform converts a state from the computational basis to the Fourier basis

$$QFT|\psi\rangle = |\psi\rangle_{QFT} = |\psi\rangle$$

Quantum Fourier Transformation Algorithm Explanation

- Skipping over a lot of kinda complex math here, math to come in next slide show
- Main point $\longrightarrow$ Look at the DFT equation and the controlled phase gate
- The algorithm encodes the values of the qubits into the *phase* of the quantum state

Quantum Fourier Transformation Algorithm Explanation

- Algorithm is based on Hadamard gates (which but qubits into superpositions) and controlled phase gates which apply between every pair of qubits
  - Relative phase shift because it depends on where the qubits are in relation to each other
- Placing the qubits into superpositions and then applying successive relative phase gates will enhance patterns present in the data

Quantum Fourier Transformation Algorithm

- To the highest indexed qubit, apply a Hadamard gate
- Apply a CPhase gate from the highest indexed qubit (target) to the second highest indexed qubit (control) with $\phi = \frac{\pi}{2^k}$, where k is the difference in the qubit indices
- Do the same for the highest indexed qubit (target) and the third highest indexed qubit (control), repeat until a CPhase gate exists between the highest indexed qubit and all other qubits

Quantum Fourier Transformation Algorithm

- To the second highest indexed qubit, apply a Hadamard gate
- Apply a CPhase gate from the second highest indexed qubit (target) to the third highest indexed qubit (control) with $\phi = \frac{\pi}{2^k}$, where k is the difference in the qubit indices
- Do the same for the second highest indexed qubit (target) and the fourth highest indexed qubit (control), repeat until a CPhase gate exists between the second highest indexed qubit and all other qubits

Quantum Fourier Transformation Algorithm

- Repeat the prior pattern until all qubits have a Hadamard gate and CPhase gates exist between every pair of qubits
- Finally, reverse the order of all qubits (this is because of how Qiskit orders the qubits)

One-Qubit Example

- A quantum Fourier transformation is applied to a single qubit with a Hadamard gate (remember the N=2 DFT matrix)
- **To the board!**
- **Now to Jupyter!**

Two-Qubit Example

- For the two-qubit quantum Fourier we need two Hadamard gates and one controlled phase gate.
- **To the board!**
- **Now to Jupyter!**

Three-Qubit Example

- Three Hadamard Gates, three CPhase gates...!
- **To the board!**
- **Now to Jupyter!**

Generalization

- How can we generalize this...maybe with recursion?
- **To the board!**
- **Now to Jupyter!**