PHY256 Lecture notes: Introducing Quantum Algorithms

A. C. Quillen

May 3, 2021

Contents

1 Acronyms for Complexity

We list some terminology.

1.1 P and NP

- A **decision problem** is a problem that gives a yes or no answer. The output is Boolean.
- **P**. Decision problems solved by polynomial size circuit families. This is the general class of questions for which some algorithm can provide an answer in polynomial time.
- **NP**. The class of questions for which an answer can be **verified** in polynomial time.

Can problems that can be verified in polynomial time (**NP**) can also be solved in polynomial time (**P**)? If it turns out that $\mathbf{P} \neq \mathbf{NP}$, which is widely believed, it would mean that there are problems in **NP** that can be **verified** in polynomial time but not **solved** in polynomial time.

1.2 CIRCUT-SAT, PSPACE, BPP, BQP

- CIRCUIT-SAT. You are given a circuit C. You want to find out if there is an x (string of bits) such that $C(x) = 1$. CIRCUIT-SAT $\in$ **NP** as C is a verifying function that can be quickly computed.
- **NP-complete**. A problem **A** in **NP** is **NP-complete** if every problem in **NP** can be reduced to problem **A**. CIRCUIT-SAT is **NP** complete.
- **PSPACE**. Problems that can be solved in polynomial space but may require exponential time.
- **BPP**. Bounded-error probabilistic polynomial time. It is guaranteed to run in polynomial time and has a probability of less than 1/2 or 1/3 of getting the wrong answer.
- **BQP**. Bounded error quantum polynomial time. The class of problems that can be decided with high probability by polynomial size quantum circuits.

- Quantum supremacy. A computation that can be done on a quantum computer with fewer operations than on a classical computer demonstrates quantum supremacy.

Nothing is as yet known about the relation between **BQP** and **NP** or **P**.
It is not known whether **BPP** $\subseteq$ **BQP** is a proper inclusion.
It is an open question as to whether **BPP = PSPACE**.

2 Black box problems

The goal of this section is to show how entanglement and superposition can be used on quantum computers to speed up certain types of calculations.

2.1 Deutsch's problem

Consider a Boolean function $f(x)$ on a single bit with $x \in \{0, 1\}$ that gives an output also $\in \{0, 1\}$. There are 4 possibilities for the function

$$f(x) = 0 \qquad f(x) = 1 \qquad f(x) = x \qquad f(x) = \text{NOT } x.$$

The left two are **constant** Boolean functions as they always give the same out bit. The right two possibilities are called **balanced** functions as they give both 0 and 1 as outputs. Classically you would have to call the function *twice* to figure out if the function is constant or balanced. However it is possible to design a quantum circuit that uses interference and a *single* function call to determine whether the function is constant or balanced. The circuit is illustrated in Figure 1. The unitary operation U_f takes

$$U_f : \qquad |xy\rangle \rightarrow |x, y + f(x)\rangle \tag{1}$$

and is only called a single time in the circuit.

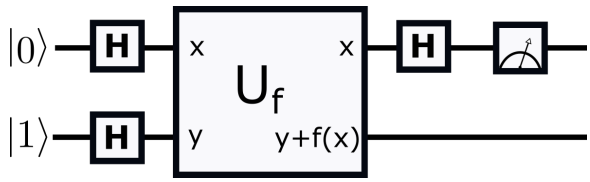

Figure 1: The Deutsch algorithm is the following quantum circuit applied to an initial state of $|01\rangle$. The U_f transformation is $|xy\rangle \rightarrow |x, y + f(x)\rangle$ where $x, y \in \{0, 1\}$ and the $+$ is mod 2. Here $f()$ is a Boolean function (returning 0 or 1). The goal is to determine if $f(x)$ is constant (with $f(0) = f(1)$) or balanced (with $f(0) = \text{NOT } f(1)$) with a single call of the operator U_f.

To illustrate this computationally rather than analytically we need to implement the unitary operation U_f. We compute the following for U_f

$$\begin{aligned}
f(x) = 0 \qquad & |00\rangle \to |00\rangle, |01\rangle \to |01\rangle, |10\rangle \to |10\rangle, |11\rangle \to |11\rangle \\
f(x) = 1 \qquad & |00\rangle \to |01\rangle, |01\rangle \to |00\rangle, |10\rangle \to |11\rangle, |11\rangle \to |10\rangle \\
f(x) = x \qquad & |00\rangle \to |00\rangle, |01\rangle \to |01\rangle, |10\rangle \to |11\rangle, |11\rangle \to |10\rangle \\
f(x) = \bar{x} \qquad & |00\rangle \to |01\rangle, |01\rangle \to |00\rangle, |10\rangle \to |10\rangle, |11\rangle \to |11\rangle .
\end{aligned}$$

The unitary operation U_f depends on the form of f

$$U_{f(x)=0} = \mathbf{I}$$

$$U_{f(x)=1} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} = \mathbf{I} \otimes \sigma_x$$

$$U_{f(x)=x} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} = \text{CNOT}(\text{control} = 0, \text{target} = 1)$$

This one is equivalent to the CNOT gate with control bit the first one and target bit the second one. It is also $|xy\rangle \to |x, y + x\rangle$.

$$U_{f(x)=\bar{x}} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \text{CNOT}(\text{control} = 0, \text{target} = 1) * (\mathbf{I} \otimes \sigma_x)$$

This one flips the second bit only if the first bit is 0. It is also $|xy\rangle \to |x, y + \text{ NOT } x\rangle$.

Let's mimic the operation of the circuit shown in Figure 1. Starting with initial state $|01\rangle$ we apply $\mathbf{H} \otimes \mathbf{H}$

$$\begin{aligned}
\mathbf{H} \otimes \mathbf{H} |01\rangle &= \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \\
&= \frac{1}{2}(|00\rangle - |01\rangle + |10\rangle - |11\rangle)
\end{aligned}$$

Now apply $U_f = |xy\rangle \to |x, y + f(x)\rangle$

$$\begin{aligned}
|00\rangle &\to |0, 0 + f(0)\rangle = |0, f(0)\rangle \\
|01\rangle &\to |0, 1 + f(0)\rangle = |0, \text{NOT } f(0)\rangle \\
|10\rangle &\to |1, 0 + f(1)\rangle = |1, f(1)\rangle \\
|11\rangle &\to |1, 1 + f(1)\rangle = |1, \text{NOT f}(1)\rangle .
\end{aligned}$$

The result is

$$\mathbf{U}_f \mathbf{H} \otimes \mathbf{H} \left|01\right\rangle = \frac{1}{2}(\left|0, f(0)\right\rangle - \left|0, \text{NOT f(0)}\right\rangle + \left|1, f(1)\right\rangle - \left|1, \text{NOT } f(1)\right\rangle). \tag{2}$$

Now we apply the Hadamard operation to the first bit. The Hadamard operator takes

$$\begin{aligned}
\left|0, f(0)\right\rangle &\rightarrow \frac{1}{\sqrt{2}}(\left|0, f(0)\right\rangle + \left|1, f(0)\right\rangle) \\
\left|0, \text{NOT } f(0)\right\rangle &\rightarrow \frac{1}{\sqrt{2}}(\left|0, \text{NOT } f(0)\right\rangle + \left|1, \text{NOT } f(0)\right\rangle) \\
\left|1, f(1)\right\rangle &\rightarrow \frac{1}{\sqrt{2}}(\left|0, f(1)\right\rangle - \left|1, f(1)\right\rangle) \\
\left|1, \text{NOT } f(1)\right\rangle &\rightarrow \frac{1}{\sqrt{2}}(\left|0, \text{NOT } f(1)\right\rangle - \left|1, \text{NOT } f(1)\right\rangle)
\end{aligned}$$

The operation of the Hadamard to the first qubit on the state in equation 2 gives a final or output state

$$\begin{aligned}
\left|\text{final}\right\rangle &= (\mathbf{H} \otimes \mathbf{I}) * \mathbf{U}_f * (\mathbf{H} \otimes \mathbf{H}) \left|01\right\rangle \\
&= \frac{1}{\sqrt{8}}(\left|0, f(0)\right\rangle - \left|0, \text{NOT } f(0)\right\rangle + \left|0, f(1)\right\rangle - \left|0, \text{NOT } f(1)\right\rangle + \\
&\quad \left|1, f(0)\right\rangle - \left|1, \text{NOT } f(0)\right\rangle - \left|1, f(1)\right\rangle + \left|1, \text{NOT } f(1)\right\rangle .
\end{aligned}$$

If $f(0)$ is equal to $f(1)$, and f is a constant function, then the final state evaluates to

$$\left|\text{final}\right\rangle_{\text{constant}} = \frac{1}{\sqrt{2}}(\left|0, f(0)\right\rangle - \left|0, \text{NOT } f(0)\right\rangle)$$

We notice that the first bit is now exclusively in the 0 state!

If $f(0)$ is opposite to $f(1)$, and f is balanced, then the final state is

$$\left|\text{final}\right\rangle_{\text{balanced}} = \frac{1}{\sqrt{2}}(\left|1, f(0)\right\rangle - \left|1, \text{NOT } f(0)\right\rangle)$$

We notice that the first bit is now exclusively in the 1 state! Measurement of the first bit can determine whether the function is constant or balanced.

This is the Deutsch algorithm. It is possible to measure whether the Boolean function $f()$ is constant or balanced using a single quantum call of the function $f()$.

2.2 The N-bit Hadamard transformation

In the Deutsch problem we applied the Hadamard to two qubits. The Hadamard operation applied to many or all qubits is a common element of many quantum algorithms.

We consider a system of N qubits. A number of algorithms use the N-bit Hadamard transformation which is the tensor product of a Hadamard transformation for each bit

$$\mathbf{W}_N \equiv \mathbf{H} \otimes \mathbf{H} \otimes \mathbf{H} \otimes ... \otimes \mathbf{H}$$

This is sometimes called the Walsh-Hadamard transformation.

Let's do the operation on 3 qubits starting with $|\psi\rangle = |000\rangle$,

$$\begin{aligned}
\mathbf{W}_3 |000\rangle &= \mathbf{H} \otimes \mathbf{H} \otimes \mathbf{H} |000\rangle \\
&= \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \\
&= \frac{1}{2^{3/2}}(|000\rangle + |001\rangle + |010\rangle + |011\rangle + |100\rangle + |101\rangle + |110\rangle + |111\rangle) \\
&= \frac{1}{2^{3/2}} \sum_{x=000}^{111} |x\rangle \, .
\end{aligned}$$

Here x is a binary string that has length 3.

Starting with wave vector $|\psi\rangle = |0000....0\rangle$ what does the N-bit Hadamard transformation do?

$$\begin{aligned}
\mathbf{W}_N |000...0\rangle &= \mathbf{H} \otimes \mathbf{H} \otimes \mathbf{H} \otimes \mathbf{H} \otimes ... \otimes \mathbf{H} |0000....0\rangle \\
&= \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes ... \otimes \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \\
&= \frac{1}{2^{N/2}}(|0000...\rangle + |1000..\rangle + |0100...\rangle + |1100..\rangle + |1010..\rangle + ...) \\
&= \frac{1}{2^{N/2}} \sum_{x=0}^{2^N-1} |x\rangle \, .
\end{aligned}$$

Here x is a binary string that has length N. The binary string x is a string of digits $x_0 x_1 x_2 ....$ where each digit $x_a \in \{0, 1\}$. Sometimes people write digit $x_a \in \mathbb{Z}_2$ and $x \in (\mathbb{Z}_2)^N$. If we think of x as a number in base 2 then x is also an integer $\in \{0, 1, 2, ..., 2^{N-1}\}$. With N qubits, a basis for wave vector is also given by the binary strings that have N digits.

What if we apply the $\mathbf{W}_3$ transformation to a different state vector? Again starting

with the 3-bit version of $\mathbf{W}$

$$\begin{aligned}
\mathbf{W}_3 |001\rangle &= \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \\
&= \frac{1}{2^{3/2}}(|000\rangle - |001\rangle + |010\rangle - |011\rangle + |100\rangle - |101\rangle + |110\rangle - |111\rangle) \\
&= \frac{1}{2^{3/2}} \sum_{i_0=0}^{1} \sum_{i_1=0}^{1} \sum_{i_2=0}^{1} |i_0 i_1 i_2\rangle \, (-1)^{i_2} \\
&= \frac{1}{2^{3/2}} \sum_{x=0}^{2^3-1} |x\rangle \, (-1)^{i_2}.
\end{aligned}$$

In the last step x is a 3 bit string of bits. With the state

$$\begin{aligned}
\mathbf{W}_3 |011\rangle &= \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \\
&= \frac{1}{2^{3/2}}(|000\rangle - |001\rangle - |010\rangle + |011\rangle + |100\rangle - |101\rangle - |110\rangle + |111\rangle) \\
&= \frac{1}{2^{3/2}} \sum_{i_0=0}^{1} \sum_{i_1=0}^{1} \sum_{i_2=0}^{1} |i_0 i_1 i_2\rangle \, (-1)^{i_1+i_2} \\
&= \frac{1}{2^{3/2}} \sum_{x=0}^{2^3-1} |x\rangle \, (-1)^{i_1+i_2}.
\end{aligned}$$

We see a pattern that we can write in terms of the number of common bits in x and in the wave vector upon which $\mathbf{W}$ operates that are 1.

It is convenient to define a new symbol which we write as $x \cdot y$ where x, y are both N bit strings. The product $x \cdot y$ is the number of digits (mod 2) that are both 1 in both strings. If x has bits $x_0 x_1 x_2$ and y has bits $y_0 y_1 y_2$

$$\begin{aligned}
x \cdot y &= (x_0 x_1 x_2) \cdot (y_0 y_1 y_2) \\
&= (x_0 \wedge y_0) \oplus (y_1 \wedge y_1) \oplus (x_2 \wedge y_2) \\
&= x_0 y_0 + y_1 y_1 + x_2 y_2.
\end{aligned}$$

Let's check that this works: For the first case with $|\psi\rangle = |001\rangle$, the expression $x \cdot 001$ is only 1 if the third bit in x is 1. In the second case with $|\psi\rangle = |011\rangle$, the expression $x \cdot 011$ is only 1 if one of the second and third bits in x are 1 but not both. This is consistent with the examples we did above!

As the power of -1 can only be one of two values, 1 or -1, a general expression for the N bit Hadamard

$$\mathbf{W}_N |y\rangle = \mathbf{H}^N |y\rangle = \frac{1}{2^{N/2}} \sum_{x=0}^{2^N-1} (-1)^{x \cdot y} |x\rangle \, . \tag{3}$$

where x, y are both N bit binary strings.

2.3 Deutsch-Jozca problem

A more general version of the Deutsch problem with function $f(x)$ that depends on x a binary string of length N (or a series of bits) is known as the **Deutsch-Jozca algorithm**. Previously we looked at the unitary operation (equation 1)

$$U_f : \qquad |xy\rangle \to |x, y + f(x)\rangle \qquad x, y \in \{0, 1\} \tag{4}$$

with $f(x)$ a Boolean function. We can more consider the same expression but more generally with $x \in (\mathbb{Z}_2)^N = \{0, 1\}^N$ (strings of bits of length N) and with y still a single bit. The function $f(x)$ takes a string of bits of length N to $\{0, 1\}$ so its output is Boolean.

$$f : \qquad \{0, 1\}^N \to \{0, 1\}$$

We restrict the form of f and **assert** that it is either **constant** or **balanced**. If f is constant then $f(x) = 0$ for all x or $f(x) = 1$ for all x. If f is balanced then $f(x) = 0$ for half of all possible x string values. There are functions on N bits with $N > 1$ that are neither constant nor balanced so this assertion is quite restrictive.

Again we can ask is f constant or is f balanced? And can we find out with a single operation of U_f with

$$U_f : \qquad |xy\rangle \to |x, y + f(x)\rangle \qquad x \in \{0, 1\}^N, y \in \{0, 1\}. \tag{5}$$

To find out if f is constant or balance we use a quantum circuit that is similar to that in Figure 1 for the Deutsch problem and that is shown in Figure 2. The Hadamard on the $|0\rangle$ state used in the Deutsch problem is replaced by the N-bit Hadamard on $|0\rangle^N$.

The action of the circuit is

$$(H^N \otimes H)\,|0\rangle^N \otimes |1\rangle = \frac{1}{2^{N/2}} \sum_{x=0}^{2^N-1} |x\rangle \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

$$U_f(H^N \otimes H)\,|0\rangle^N \otimes |1\rangle = \frac{1}{2^{N/2+1/2}} \sum_{x=0}^{2^N-1} (|x\rangle \otimes |f(x)\rangle - |x\rangle \otimes |1 + f(x)\rangle)$$

$$(H^N \otimes I)U_f(H^N \otimes H)\,|0\rangle^N \otimes |1\rangle = \frac{1}{2^{N+1/2}} \sum_{x=0}^{2^N-1} \sum_{y=0}^{2^N-1} (-1)^{xy}(|y\rangle \otimes |f(x)\rangle - |y\rangle \otimes |1 + f(x)\rangle).$$

If $f(x) = 0$ then $|f(x)\rangle - |1 + f(x)\rangle = |0\rangle - |1\rangle$.
If $f(x) = 1$ then $|f(x)\rangle - |1 + f(x)\rangle = |1\rangle - |0\rangle = (-1)^{f(x)} \times (|0\rangle - |1\rangle)$.
In general

$$|f(x)\rangle - |1 + f(x)\rangle = (-1)^{f(x)}(|0\rangle - |1\rangle)$$

and the final state is

$$|\psi\rangle_{\text{final}} = \frac{1}{2^{N+1/2}} \sum_{x=0}^{2^N-1} \sum_{y=0}^{2^N-1} (-1)^{xy} (-1)^{f(x)} (|y\rangle \otimes (|0\rangle - |1\rangle)) \tag{6}$$

If the function is constant let $f(x) = c$ where $c \in \{0, 1\}$ and the final state is

$$|\psi\rangle_{\text{final}} = \frac{1}{2^{N+1/2}} \sum_{x=0}^{2^N-1} \sum_{y=0}^{2^N-1} (-1)^{xy} (-1)^{c} (|y\rangle \otimes (|0\rangle - |1\rangle)) \qquad \text{for } f \text{ constant}$$

where x and y are N-bit strings. We need to compute the sum

$$\frac{1}{2^N} \sum_{x=0}^{2^N-1} (-1)^{xy}.$$

For any non-zero y string we are going to get as many +1 as -1 in the sum so this will be zero unless $y = 00000...0$. If $y = 0000...0$ then the sum gives 1. Hence

$$\frac{1}{2^N} \sum_{x=0}^{2^N-1} (-1)^{xy} = \delta_{y0}. \tag{7}$$

The final wave function is

$$|\psi\rangle_{\text{final}} = |0\rangle^N \otimes \frac{1}{\sqrt{2}} (-1)^{f(0)} (|0\rangle - |1\rangle) \quad \text{for } f() \text{ constant.}$$

Measurement of the the first N bits should always give a series of zeros.

We now consider the case of $f(x)$ balanced

$$|\psi\rangle_{\text{final}} = \frac{1}{2^{N+1/2}} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} (-1)^{xy} (-1)^{f(x)} (|y\rangle \otimes (|0\rangle - |1\rangle)).$$

We need to compute the sum

$$\frac{1}{2^{N+1/2}} \sum_{x=0}^{N-1} (-1)^{xy+f(x)}$$

Consider the case of $y = 0000...0$.

$$\langle 0|^N |\psi\rangle_{\text{final}} = \frac{1}{2^{N+1/2}} \sum_{x=0}^{N-1} (-1)^{f(x)} (|0\rangle - |1\rangle)$$

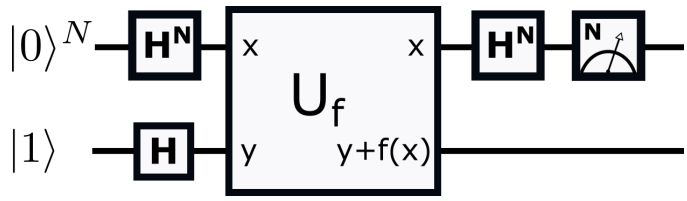

Figure 2: The Deutsch-Jozca algorithm is the following quantum circuit applied to an initial state of $|0\rangle^N \otimes |1\rangle$. Here the total number of qubits is $N+1$. The U_f transformation is $|xy\rangle \rightarrow |x, y + f(x)\rangle$ where $x \in (\{0,1)\})^N$, $y \in \{0,1\}$ and the $+$ is mod 2. Here $f()$ is a Boolean function (returning 0 or 1) that is either constant or balanced. The goal is to determine if $f()$ is constant or balanced with a single call of the operator U_f.

We need to compute

$$\frac{1}{2^N} \sum_{x=0}^{N-1} (-1)^{f(x)},$$

but if $f(x)$ is balanced then this sum is zero. This implies that

$$\langle 0|^N |\psi\rangle_{\text{final}} = 0.$$

When the measurement of the first N bits is done they will not all be zero.

In summary, if the function is constant then the first N bits measured will all be zero. Any other result implies that the function is balanced.

Is there an advantage in doing the quantum calculation? In a classical setting 2^N queries of the function are required to determine if the function is constant, but here we used mixed states to determine if the function is constant with a single query of the function. After a few queries we would be unlikely to get the same value if the function were actually balanced. Classically the computation is not considered **hard** because a polynomial number of queries can give an exponentially good estimate for whether the function is constant or balanced.

2.4 What is an oracle?

The Deutsch and Deutsch-Jozca problems are **oracle** problems. The function is like a **black box** and you can ask it questions (the function queries). The idea is that if you ask the oracle enough questions, you can determine what is in the box. In the quantum setting, the unitary operation U_f serves as the quantum oracle. The circuit **complexity** is the number of calls (querries) made to the oracle to determine how the oracle functions.

Additional quantum oracle problems are the Bernstein-Vazirani problem, Simon's problem and Grover's problem.

2.5 Non-invertible functions and quantum parallelism

For a classical computer one evaluates a function of a series of bits. For a quantum computer one evaluates a function on a series of qubits that can be in quantum superpositions of states. Calling the function a single time gives what is called **quantum parallelism** and we have been seeing this exploited in the Deutsch and Deutsch-Jozca algorithms.

Quantum gates are unitary operations which are invertible. How do we evaluate functions that are not necessarily invertible? How do we write irreversible functions in terms of unitary or invertible operations? Consider a function $f(\mathbf{x})$ of N bits that returns 0 or 1, and define a function of $N+1$ bits

$$F(\mathbf{x}, y) = (\mathbf{x}, y + f(\mathbf{x}))$$

with $\mathbf{x} \in \{0,1\}^N$, $f(x) \in \{0,1\}$, $y \in \{0,1\}$. Here the plus is mod 2 or equivalent to the XOR. Because we have $\mathbf{x}$ in the output we can compute $f(\mathbf{x})$, subtract it off the last bit and recover y. The transformation is invertible. However any Boolean function can be computed, including those that are not invertible. Quantum computing requires extra information to be retained during the calculation.

2.6 Bernstein-Vazirani algorithm

We once again are given an oracle function

$$f: \qquad \{0,1\}^N \to \{0,1\}.$$

We assert that $f(x) = s \cdot x \pmod 2$ for some secret N bit string s. Here

$$s \cdot x = s_0 x_0 + s_1 x_1 + ... + s_{N-1} x_{N-1} \bmod 2.$$

The problem is to find s with the fewest oracle queries.

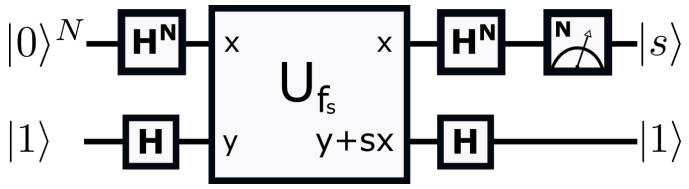

Figure 3: The Bernstein-Vazirani problem is solved with this quantum circuit applied to an initial state of $|0\rangle^N \otimes |1\rangle$. Here the total number of qubits is $N+1$. The U_f transformation is $|xy\rangle \to |x, y + f(x)\rangle$ where $x \in (\{0,1)\})^N$, $y \in \{0,1\}$ and the + is mod 2. The function $f(x) = sx$ for a mystery N-bit binary string s. The goal is to determine the string s with a single call of the operator U_f.

Our oracle does this

$$U_f \left|x, y\right\rangle = \left|x, y + sx\right\rangle .$$

Classically to find s you must query the function N times

$$\begin{aligned} f(1000..0) &= s_0 \\ f(0100..0) &= s_1 \\ f(0010..0) &= s_2 \\ f(0001..0) &= s_3 \\ :::: &: \\ f(0000..(N-1)) &= s_{N-1} \end{aligned}$$

No classical algorithm has fewer queries, since each query only provides one bit of information about the mystery string s.

Not surprisingly the Bernstein-Vazirani algorithm is similar to the Deutsch-Jozca problem. Start with $\left|0\right\rangle^N$, then apply the N-bit Hadamard operator H^N then query with U_f, then apply the N-bit Hadamard operator again, the measure all bits. The result is s.

The action of the circuit is

$$\begin{aligned} (H^N \otimes H) \left|0\right\rangle^N \otimes \left|1\right\rangle &= \frac{1}{2^{N/2}} \sum_{x=0}^{2^N-1} \left|x\right\rangle \otimes \frac{1}{\sqrt{2}}(\left|0\right\rangle - \left|1\right\rangle) \\ U_f(H^N \otimes H) \left|0\right\rangle^N \otimes \left|1\right\rangle &= \frac{1}{2^{N/2+1/2}} \sum_{x=0}^{2^N-1} (\left|x\right\rangle \otimes \left|f(x)\right\rangle - \left|x\right\rangle \otimes \left|1 + f(x)\right\rangle) \\ &= \frac{1}{2^{N/2+1/2}} \sum_{x=0}^{2^N-1} (\left|x\right\rangle \otimes \left|sx\right\rangle - \left|x\right\rangle \otimes \left|1 + sx\right\rangle) \\ &= \frac{1}{2^{N/2+1/2}} \sum_{x=0}^{2^N-1} (-1)^{sx} \left|x\right\rangle \otimes (\left|0\right\rangle - \left|1\right\rangle) \\ (I \otimes H) U_f(H^N \otimes H) \left|0\right\rangle^N \otimes \left|1\right\rangle &= \frac{1}{2^{N/2}} \sum_{x=0}^{2^N-1} (-1)^{sx} \left|x\right\rangle \otimes \left|1\right\rangle \end{aligned}$$

Lastly we use equation 3 to apply the last operation of H^N

$$(H^N \otimes H) U_f (H^N \otimes H) \left|0\right\rangle^N \otimes \left|1\right\rangle = \frac{1}{2^N} \sum_{x=0}^{2^N-1} \sum_{y=0}^{2^N-1} (-1)^{sx} (-1)^{xy} \left|y\right\rangle \otimes \left|1\right\rangle \tag{8}$$

We can write $(-1)^{sx+xy} = (-1)^{(s+y)x}$. Then as we did previously (see equation 7) when

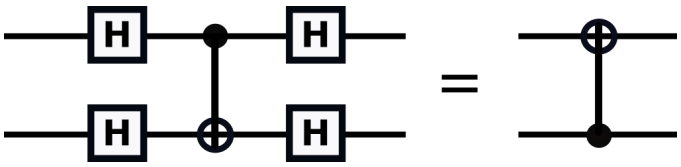

Figure 4: These two circuits are equivalent.

computing the Deutsch Jozca algorithm, the sum

$$\frac{1}{2^N}\sum_{x=0}^{2^N-1}(-1)^{qx}=\delta_{q0}.$$

Our sum in equation 8 is zero unless $s+y=0$. The only term that remains in equation 8 has $y=s$ so

$$|\psi\rangle_{\text{final}}=(H^N\otimes H)U_f(H^N\otimes H)\,|0\rangle^N\otimes|1\rangle=|s\rangle\otimes|1\rangle\,.$$

We measure the first N bits and we recover the mystery string s! Amazingly this worked with only one query of the quantum oracle function U_f!

Why does this work? We used quantum parallelism and superposition to remove all incorrect results. Another way to look at it is to notice the 4 Hadamard transformations. 4 Hadamard transformations on a CNOT reverse the control bit and target bits (see Figure 4). The circuit behaves as if it were applying a CNOT for every bit of the mystery string s.

3 The Quantum Fourier Transform

The Quantum Fourier transform is used in a number of algorithms such as the Shor algorithm.

3.1 The Discrete Fourier Transform

The Discrete Fourier Transform is used on an evenly spaced set of data points. A particularly efficient (in both numbers of operations and memory usage) is the Fast Fourier Transform (FFT) which is done on $N=2^n$ data samples, where the number of samples is a power of 2.

We take x_j to be our N data samples with index j going from 0 to $N-1$. The numbers x_j can be complex. The result of the Discrete Fourier Transform is another sequence of complex numbers

$$X_k=\frac{1}{\sqrt{N}}\sum_{j=0}^{N-1}x_j e^{\frac{2\pi i}{N}kj} \tag{9}$$

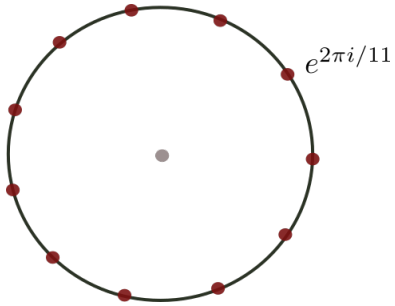

Figure 5: By symmetry, the roots of unity on the complex plane sum to zero.

with index k also ranging from 0 to $N-1$. The inverse transform is

$$x_j = \frac{1}{\sqrt{N}} \sum_{l=0}^{N-1} X_l e^{-\frac{2\pi i}{N} lj}. \tag{10}$$

The normalization of these two transforms is need not both involve $\sqrt{N}$. Some times people chose to define the transform without the factor of $\frac{1}{\sqrt{N}}$ and then the inverse transform has a factor of $1/N$ in it. Also sometimes the signs of the exponents are flipped (the transform has a minus sign and the inverse transform has a plus sign instead of what is written here).

It is useful to compute a few sums involving complex roots of unity. The numbers $e^{\frac{2\pi}{N} j}$ with integer $j \in [0, N-1]$, are N evenly spaced points on the unit circle on the complex plane. The points are symmetrically distributed about the real and imaginary axes on the complex plane, as showin in Figure 5, so the sum

$$\sum_{j=0}^{N-1} e^{\frac{2\pi i}{N} j} = 0.$$

With integer $q \in [0, N-1]$, the sum

$$\sum_{j=0}^{N-1} e^{\frac{2\pi i}{N} qj} = \begin{cases} 0 & \text{if} \quad q \neq 0 \\ N & \text{if} \quad q = 0 \end{cases}. \tag{11}$$

We can check that inverse transform is an inverse transform by inserting the sum for

x_j inside that for X_k

$$\begin{aligned} X_k &= \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j e^{\frac{2\pi i}{N} kj} \\ &= \frac{1}{N} \sum_{j=0}^{N-1} \sum_{l=0}^{N-1} X_l e^{-\frac{2\pi i}{N} lj} e^{\frac{2\pi i}{N} kj} \\ &= \frac{1}{N} \sum_{l=0}^{N-1} X_l \sum_{j=0}^{N-1} e^{\frac{2\pi i}{N}(k-l)j} \\ &= \frac{1}{N} \sum_{l=0}^{N-1} X_l \ N \ \delta_{lk} \\ &= X_k. \end{aligned}$$

In the second to last step we have used equation 11.

Let's look again at the definition of the discrete Fourier transform in equation 9 which is repeated here

$$X_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j e^{\frac{2\pi i}{N} kj}. \tag{12}$$

Notice that $e^{\frac{2\pi i}{N}}$ is a root of unity. In other words $(e^{\frac{2\pi i}{N}})^N = 1$. Let

$$\omega_N \equiv e^{\frac{2\pi i}{N}}. \tag{13}$$

The discrete Fourier transform can be written as

$$\begin{pmatrix} X_0 \\ X_1 \\ X_2 \\ ... \\ X_{N-1} \end{pmatrix} = \frac{1}{\sqrt{N}} \begin{pmatrix} 1 & 1 & 1 & 1 & ... & 1 \\ 1 & \omega_N & \omega_N^2 & \omega_N^3 & ... & \omega_N^{N-1} \\ 1 & \omega_N^2 & \omega_N^4 & \omega_N^6 & ... & \omega_N^{N-2} \\ 1 & \omega_N^3 & \omega_N^6 & \omega_N^9 & ... & \omega_N^{N-3} \\ : & : & : & : & : & : \\ 1 & \omega_N^{N-1} & \omega_N^{N-2} & \omega_N^{N-3} & ... & \omega_N \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ ... \\ x_{N-1} \end{pmatrix}. \tag{14}$$

I have simplified using $(\omega_N)^N = 1$. For example, the second row contains multiples of ω_N. The third row contains multiples of ω_N^2. The rightmost term of this row is equivalent to $(\omega_N)^{2(N-1)} = (\omega_N)^{-2} = (\omega_N)^{N-2}$.

Let's look at the matrix in equation 14

$$U_{DFT}^{(N)} = \frac{1}{\sqrt{N}} \begin{pmatrix} 1 & 1 & 1 & 1 & ... & 1 \\ 1 & \omega_N & \omega_N^2 & \omega_N^3 & ... & \omega_N^{N-1} \\ 1 & \omega_N^2 & \omega_N^4 & \omega_N^6 & ... & \omega_N^{N-2} \\ 1 & \omega_N^3 & \omega_N^6 & \omega_N^9 & ... & \omega_N^{N-3} \\ : & : & : & : & : & : \\ 1 & \omega_N^{N-1} & \omega_N^{N-2} & \omega_N^{N-3} & ... & \omega_N \end{pmatrix}. \tag{15}$$

Equation 11 implies that the rows of $U_{DFT}^{(N)}$ are orthogonal and its columns are also orthogonal. The matrix is a unitary transformation.

The $N \times N$ square matrix $U_{DFT}^{(N)}$ has indices

$$(U_{DFT}^{(N)})_{ij} = \frac{1}{\sqrt{N}} \omega_N^{ij}. \tag{16}$$

For $N = 8$ the transformation matrix is

$$U_{DFT}^{(8)} = \frac{1}{\sqrt{8}} \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & \omega^1 & \omega^2 & \omega^3 & \omega^4 & \omega^5 & \omega^6 & \omega^7 \\ 1 & \omega^2 & \omega^4 & \omega^6 & 1 & \omega^2 & \omega^4 & \omega^6 \\ 1 & \omega^3 & \omega^6 & \omega^1 & \omega^4 & \omega^7 & \omega^2 & \omega^5 \\ 1 & \omega^4 & 1 & \omega^4 & 1 & \omega^4 & 1 & \omega^4 \\ 1 & \omega^5 & \omega^2 & \omega^7 & \omega^4 & \omega^1 & \omega^6 & \omega^3 \\ 1 & \omega^6 & \omega^4 & \omega^2 & 1 & \omega^6 & \omega^4 & \omega^2 \\ 1 & \omega^7 & \omega^6 & \omega^5 & \omega^4 & \omega^3 & \omega^2 & \omega^1 \end{pmatrix} \qquad \text{with} \qquad \omega = e^{\frac{2\pi i}{8}}. \tag{17}$$

If N is a power of 2 then there is a recursive way of generating it that is the heart of the Fast Fourier Transform.

If $N = 2^n$, then $(\omega_N)^{N/2} = -1$. The matrix in equation 14 can be rearranged (columns can be reordered) and written in terms of a $N/2 \times N/2$ diagonal frequency matrix, some factors of -1 and with matrices that reorder the states. Because the iterative steps involve powers of 2, we use an index k to give us $2^k \times 2^k$ matrices. We define a complex root of 1

$$\omega_k \equiv e^{\frac{2\pi i}{k}}. \tag{18}$$

It is convenient to define

$$I^{(k)} \equiv 2^k \times 2^k \text{ identity matrix.}$$

We define the $2^k \times 2^k$ diagonal matrix

$$D^{(k)} \equiv \text{diag}(1, \omega_{k+1}, \omega_{k+1}^2, \omega_{k+1}^3, ..., \omega_{k+1}^{2^k-1}) \tag{19}$$

A $2^k \times 2^k$ matrix that helps us swap states has components

$$R_{ij}^{(k)} = \begin{cases} 1 & \text{if } 2i = j \\ 1 & \text{if } 2(i - 2^k) + 1 = j \\ 0 & \text{otherwise.} \end{cases} \tag{20}$$

The discrete Fourier transform for $N = 2^k$ states we define as

$$F^{(k)} = U_{DFT}^{(2^k)}.$$

These matrices can be used to write the $F^{(k)}$ discrete Fourier transform in terms of the $F^{(k-1)}$ transform.

$$F^{(k)} = \frac{1}{\sqrt{2}} \begin{pmatrix} I^{(k-1)} & D^{(k-1)} \\ I^{(k-1)} & -D^{(k-1)} \end{pmatrix} \begin{pmatrix} F^{(k-1)} & 0 \\ 0 & F^{(k-1)} \end{pmatrix} R^{(k)}.$$

The recursive composition makes the Discrete Fourier transform efficient. If $N = 2^n$, then $F^{(k)}$ is used n or $\log_2 N$ times. The matrix multiplications only involve N operations as the matrices are diagonal or sparse. The number of computations is $\mathrm{O}(N \log N)$.

3.2 Quantum Fourier transforms

The quantum Fourier transformation (QFT) is an important quantum subroutine. It and its generalizations are used in quantum algorithms that achieve a significant speedup over classical algorithms. It is used in Shor's algorithm which achieves BQP time for factoring integers.

We work in a N dimensional Hilbert space with dimension $N = 2^n$ that is a multiple of 2. The number of qubits is n.

We start with a wave function

$$|\psi\rangle = \sum_{x=0}^{N-1} a_x |x\rangle$$

where x are integers and the amplitudes a_x are complex numbers. The integer x written in powers of 2

$$x = x_1 2^{n-1} + x_2 2^{n-2} + x_3 2^{n-3} ... + x_{n-1} 2 + x_n 2^0$$

and could be described in terms of the string of digits

$$\text{binary string} \ : \qquad x_1, x_2, x_3, ..., x_n$$

where the digits $x_i \in \{0, 1\}$ are either 1s or 0s and the binary string $\in \{0, 1\}^n$. Notice we are indexing from 1 instead of from 0! The basis state can described in terms of these digits

$$|x\rangle = |x_1, x_2, x_3, ..., x_n\rangle$$

or in terms of the integer x. As a binary fraction

$$\frac{x}{2^n} = x_1 2^{-1} + x_2 2^{-2} + x_3 2^{-3} .....x_n 2^{-n}. \tag{21}$$

As a binary decimal

$$\frac{x}{2^n} = 0.x_1 x_2 x_3 x_4 ....x_n. \tag{22}$$

Example with $n = 5$ qubits and $N = 2^5 = 32$ possible basis states		
Integer $x = 6$	binary expansion $0 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$	binary string '00110'
Binary fraction $\frac{x}{2^5} = \frac{6}{32}$	binary expansion $\frac{0}{2} + \frac{0}{4} + \frac{1}{8} + \frac{1}{16} + \frac{0}{32}$	binary string '0.00110'
Wavefunction $\|x\rangle = \|6\rangle$	Wavefunction $\|00110\rangle$	

The Quantum Fourier Transform is a linear and unitary transformation from one wavefunction to another one

$$\text{QFT}: \qquad \sum_{x=0}^{N-1} a_x \left|x\right\rangle \rightarrow \sum_{y=0}^{N-1} b_y \left|y\right\rangle . \tag{23}$$

Here x, y are integers from 0 to $N-1$, however $\left|x\right\rangle, \left|y\right\rangle$ can either be written as integers or with x, y as binary strings. The amplitudes of the two wave functions are related by the Discrete Fourier Transform

$$\text{DFT}: \qquad b_y = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} e^{2\pi i x y/N} a_x. \tag{24}$$

The Discrete Fourier transform is the same as we discussed in section 3.1.

What is meant by the product xy in the exponential in equation 24? This is the product of two integers. It is not a bitwise operation.

The Quantum Fourier Transform (QFT) of a basis vector $\left|x\right\rangle$

$$\text{QFT}\left|x\right\rangle \equiv \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} e^{2\pi i x y/N} \left|y\right\rangle .$$

This means that the Quantum Fourier transform can be described with a matrix that is in the form of equation 15 and with components in equation 16.

3.3 3-qubit Quantum Fourier Transforms

Let's compute the Quantum Fourier transform of $|000\rangle$ for a 3 qubit system. The number of states is $N = 2^3 = 8$.

$$\begin{aligned}\mathrm{QFT}\,|000\rangle &= \frac{1}{\sqrt{8}}\sum_{y=0}^{8} e^{2\pi i 0y/8}\,|y\rangle \\ &= \frac{1}{\sqrt{8}}(|000\rangle + |001\rangle + |010\rangle + |011\rangle + |100\rangle + |101\rangle + |110\rangle + |111\rangle) \\ &= \frac{1}{\sqrt{8}}(|0\rangle + |1\rangle) \otimes (|0\rangle + |1\rangle) \otimes (|0\rangle + |1\rangle).\end{aligned}$$

Let's compute the Quantum Fourier transform of $|001\rangle$ using $\omega = e^{2\pi i/8}$

$$\begin{aligned}\mathrm{QFT}\,|001\rangle &= \frac{1}{\sqrt{8}}\sum_{y=0}^{8} e^{2\pi i 1y/8}\,|y\rangle = \frac{1}{\sqrt{8}}\sum_{y=0}^{8} \omega^y\,|y\rangle \\ &= \frac{1}{\sqrt{8}}(|000\rangle + \omega\,|001\rangle + \omega^2\,|010\rangle + \omega^3\,|011\rangle + \omega^4\,|100\rangle + \omega^5\,|101\rangle + \omega^6\,|110\rangle + \omega^7\,|111\rangle) \\ &= \frac{1}{\sqrt{8}}(|0\rangle + \omega^4\,|1\rangle) \otimes (|00\rangle + \omega\,|01\rangle + \omega^2\,|10\rangle + \omega^3\,|11\rangle) \\ &= \frac{1}{\sqrt{8}}(|0\rangle + \omega^4\,|1\rangle) \otimes (|0\rangle + \omega^2\,|1\rangle) \otimes (|0\rangle + \omega\,|1\rangle)\end{aligned}$$

Let's compute the Quantum Fourier transform of $|010\rangle$

$$\begin{aligned}\mathrm{QFT}\,|010\rangle &= \frac{1}{\sqrt{8}}\sum_{y=0}^{8} \omega^{2y}\,|y\rangle \\ &= \frac{1}{\sqrt{8}}(|000\rangle + \omega^2\,|001\rangle + \omega^4\,|010\rangle + \omega^6\,|011\rangle + |100\rangle + \omega^2\,|101\rangle + \omega^4\,|110\rangle + \omega^6\,|111\rangle) \\ &= \frac{1}{\sqrt{8}}(|0\rangle + |1\rangle) \otimes (|00\rangle + \omega^2\,|01\rangle + \omega^4\,|10\rangle + \omega^6\,|11\rangle) \\ &= \frac{1}{\sqrt{8}}(|0\rangle + |1\rangle) \otimes (|0\rangle + \omega^4\,|1\rangle) \otimes (|0\rangle + \omega^2\,|1\rangle)\end{aligned}$$

Let's compute the Quantum Fourier transform of $|011\rangle$

$$\begin{aligned}\mathrm{QFT}\,|011\rangle &= \frac{1}{\sqrt{8}}\sum_{y=0}^{8} \omega^{3y}\,|y\rangle \\ &= \frac{1}{\sqrt{8}}(|000\rangle + \omega^3\,|001\rangle + \omega^6\,|010\rangle + \omega^1\,|011\rangle + \omega^4\,|100\rangle + \omega^7\,|101\rangle + \omega^2\,|110\rangle + \omega^5\,|111\rangle) \\ &= \frac{1}{\sqrt{8}}(|0\rangle + \omega^4\,|1\rangle) \otimes (|0\rangle + \omega^6\,|1\rangle) \otimes (|0\rangle + \omega^3\,|1\rangle)\end{aligned}$$

It's not all that easy to see a pattern, but using $\omega = e^{2\pi i/8}$ our examples are consistent with the product

$$\text{QFT}\,|j_1 j_2 j_3\rangle = \frac{1}{\sqrt{8}}(|0\rangle + \omega^{4j_3}\,|1\rangle) \otimes (|0\rangle + \omega^{4j_2+2j_3}\,|1\rangle) \otimes (|0\rangle + \omega^{4j_1+2j_2+j_3}\,|1\rangle) \tag{25}$$

$$= \frac{1}{\sqrt{8}}(|0\rangle + e^{2\pi i\frac{j_3}{2}}\,|1\rangle) \otimes (|0\rangle + e^{2\pi i\left(\frac{j_2}{2}+\frac{j_3}{4}\right)}\,|1\rangle) \otimes (|0\rangle + e^{2\pi i\left(\frac{j_1}{2}+\frac{j_2}{4}+\frac{j_3}{8}\right)}\,|1\rangle). \tag{26}$$

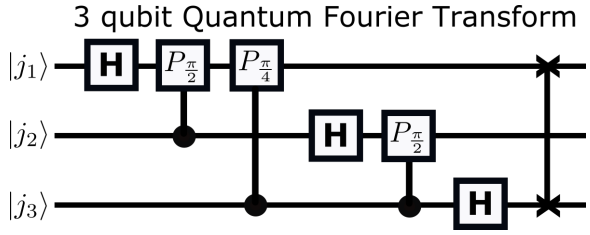

Figure 6: A circuit that computes the 3-qubit Quantum Fourier Transform. The 2-qubit gate on the right is a swap gate.

A circuit that generates the 3-qubit Quantum Fourier Transform can be done with a circuit that has the Hadamard gate, a controlled $P_{\frac{\pi}{2}}$ phase gate and a controlled $P_{\frac{\pi}{4}}$ gate. We write down the single bit versions of these gates

$$H = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix} \qquad P_{\frac{\pi}{2}} = \begin{pmatrix}1 & 0\\ 0 & i\end{pmatrix} \qquad P_{\frac{\pi}{4}} = \begin{pmatrix}1 & 0\\ 0 & e^{\frac{i\pi}{4}}\end{pmatrix}. \tag{27}$$

We first show, using brute force, that the circuit in equation 6 works to give the 3-qubit Fourier transform. Then we analytically show that the circuit is equivalent to the product formula in equation 26.

Let's show the operations in matrix form (which I checked using the python quantum toolbox `QuTip`)

$$H \otimes I \otimes I = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 0 & 0 & 0 & 1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1\\ 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0\\ 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0\\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1\end{pmatrix}$$

In QuTip `H0 = tensor(snot(),qeye(2),qeye(2))`.

$$I \otimes H \otimes I = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 \end{pmatrix}$$

In QuTip `H1 = tensor(qeye(2),snot(),qeye(2))`.

$$I \otimes I \otimes H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 \end{pmatrix}$$

In QuTip `H2 = tensor(qeye(2),qeye(2),snot())`.

The controlled phase gate $P_{\frac{\pi}{2}}$ with target top bit and control the second bit

$$\Lambda(P_{\frac{\pi}{2}}, \text{control} = 1, \text{target} = 0) = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & i & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & i \end{pmatrix}$$

In QuTip this is `cphase(np.pi/2,3,control=1,target=0)`.

The controlled phase gate $P_{\frac{\pi}{2}}$ with target the middle bit and control the third bit

$$\Lambda(P_{\frac{\pi}{2}}, \text{control} = 2, \text{target} = 1) = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & i & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & i \end{pmatrix}.$$

In QuTip this is `cphase(np.pi/2,3,control=2,target=1)`.

The controlled phase gate $P_{\frac{\pi}{4}}$ with target the first bit and control the third bit

$$\Lambda(P_{\frac{\pi}{4}}, \text{control} = 2, \text{target} = 0) = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & e^{\pi i/4} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & e^{\pi i/4} \end{pmatrix}$$

In QuTip this is `cphase(np.pi/4,3,control=2,target=0)`. Lastly we need to swap the first and last bits

$$SWAP(0,2) = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

In QuTip this is `swap(3,[0,2])`.

The entire circuit shown in Figure 6 is:

`Uw = H0*cphase(np.pi/2,3,control=1,target=0)* cphase(np.pi/4,3,control=2,target=0)*H1* cphase(np.pi/2,3,control=2,target=1)*H2*swap(3,[0,2])`

The result is

$$U_w = \frac{1}{\sqrt{8}} \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & \frac{1}{\sqrt{2}}(1+i) & i & \frac{1}{\sqrt{2}}(-1+i) & -1 & \frac{1}{\sqrt{2}}(-1-i) & -i & \frac{1}{\sqrt{2}}(1-i) \\ 1 & i & -1 & -i & 1 & i & -1 & 1 \\ 1 & \frac{1}{\sqrt{2}}(-1+i) & -i & \frac{1}{\sqrt{2}}(1+i) & -1 & \frac{1}{\sqrt{2}}(1-i) & i & \frac{1}{\sqrt{2}}(-1-i) \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & 1 \\ 1 & \frac{1}{\sqrt{2}}(-1-i) & -i & \frac{1}{\sqrt{2}}(1-i) & -1 & \frac{1}{\sqrt{2}}(1+i) & -i & \frac{1}{\sqrt{2}}(-1+i) \\ 1 & -i & -1 & 1 & 1 & -i & -1 & 1 \\ 1 & \frac{1}{\sqrt{2}}(1-i) & -i & \frac{1}{\sqrt{2}}(-1-i) & -1 & \frac{1}{\sqrt{2}}(-1+i) & i & \frac{1}{\sqrt{2}}(1+i) \end{pmatrix}.$$

We recognize that $\frac{1}{\sqrt{2}}(1+i) = e^{\pi i/4} = e^{\frac{2\pi i}{8}}$ and then it is clearer that this matrix is identical to the Discrete Fourier transformation for $N = 8 = 2^3$ with matrix $U_8 = F^{(3)}$ shown in equation 17.

We show that the circuit for the 3 qubit Fourier transform shown in Figure 6 is equivalent to the product formula shown in equation 26 which we repeat here

$$\text{QFT}\,|j_1 j_2 j_3\rangle = \frac{1}{\sqrt{8}}(|0\rangle + e^{2\pi i \frac{j_3}{2}}\,|1\rangle) \otimes (|0\rangle + e^{2\pi i\left(\frac{j_2}{2}+\frac{j_3}{4}\right)}\,|1\rangle) \otimes (|0\rangle + e^{2\pi i\left(\frac{j_1}{2}+\frac{j_2}{4}+\frac{j_3}{8}\right)}\,|1\rangle). \tag{28}$$

Starting with $|j_1 j_2 j_3\rangle$ we perform a Hadamard operation on the first qubit

$$H \otimes I \otimes I\,|j_1 j_2 j_3\rangle = \frac{1}{\sqrt{2}}(|0\rangle + (-1)^{j_1}\,|1\rangle) \otimes |j_2, j_3\rangle$$

We perform a controlled phase gate with control bit the second bit and target bit the first one. The wave vector becomes

$$\frac{1}{\sqrt{2}}(|0\rangle + (-1)^{j_1} i^{j_2}\,|1\rangle) \otimes |j_2, j_3\rangle\,.$$

We perform a controlled $P_{\frac{\pi}{4}}$ with control bit the third bit and target bit the first one. The wave vector becomes

$$\frac{1}{\sqrt{2}}(|0\rangle + (-1)^{j_1} i^{j_2} e^{i\pi j_3/4}\,|1\rangle) \otimes |j_2, j_3\rangle\,.$$

This can also be written as

$$\frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i\left(\frac{j_1}{2}+\frac{j_2}{4}+\frac{j_3}{8}\right)}\,|1\rangle) \otimes |j_2, j_3\rangle\,.$$

and we recognize an expression similar to that of last qubit in equation 28. We perform a Hadamard operation on the second qubit, the wave vector becomes

$$\frac{1}{2}(|0\rangle + e^{2\pi i\left(\frac{j_1}{2}+\frac{j_2}{4}+\frac{j_3}{8}\right)}\,|1\rangle) \otimes (|0\rangle + (-1)^{j_2}\,|1\rangle) \otimes |j_3\rangle\,.$$

We perform a controlled phase gate with control bit the third qubit and the target bit the second one. The wave vector becomes

$$\frac{1}{2}(|0\rangle + e^{2\pi i\left(\frac{j_1}{2}+\frac{j_2}{4}+\frac{j_3}{8}\right)}\,|1\rangle) \otimes (|0\rangle + (-1)^{j_2} i^{j_3}\,|1\rangle) \otimes |j_3\rangle\,.$$

This can also be written as

$$\frac{1}{2}(|0\rangle + e^{2\pi i\left(\frac{j_1}{2}+\frac{j_2}{4}+\frac{j_3}{8}\right)}\,|1\rangle) \otimes (|0\rangle + e^{2\pi i\left(\frac{j_2}{2}+\frac{j_3}{4}\right)}\,|1\rangle) \otimes |j_3\rangle\,.$$

We perform a Hadamard operation on the third qubit giving

$$\frac{1}{\sqrt{8}}(|0\rangle + e^{2\pi i\left(\frac{j_1}{2}+\frac{j_2}{4}+\frac{j_3}{8}\right)}\,|1\rangle) \otimes (|0\rangle + e^{2\pi i\left(\frac{j_2}{2}+\frac{j_3}{4}\right)}\,|1\rangle) \otimes (|0\rangle + (-1)^{j_3}\,|1\rangle).$$

Lastly we swap the first and third qubits giving

$$\frac{1}{\sqrt{8}}(|0\rangle + e^{2\pi i \frac{j_3}{2}}\,|1\rangle) \otimes (|0\rangle + e^{2\pi i\left(\frac{j_2}{2}+\frac{j_3}{4}\right)}\,|1\rangle) \otimes (|0\rangle + e^{2\pi i\left(\frac{j_1}{2}+\frac{j_2}{4}+\frac{j_3}{8}\right)}\,|1\rangle)$$

and this matches the product form of the 3-qubit QFT in equation 28.

3.4 Product representation for the Quantum Fourier Transform

There is a handy product representation for the Quantum Fourier Transform which we derive.

We repeat yet again the product form of the 3-qubit Quantum Fourier transform (previously equations 26 and 28)

$$\mathrm{QFT}\,|j_1 j_2 j_3\rangle = \frac{1}{\sqrt{8}}(|0\rangle + e^{2\pi i \frac{j_3}{2}}\,|1\rangle) \otimes (|0\rangle + e^{2\pi i \left(\frac{j_2}{2}+\frac{j_3}{4}\right)}\,|1\rangle) \otimes (|0\rangle + e^{2\pi i \left(\frac{j_1}{2}+\frac{j_2}{4}+\frac{j_3}{8}\right)}\,|1\rangle). \tag{29}$$

and recall the binary decimal form for an integer (equation 22). Using the binary decimal form for an integer we can write the product form of the 3-qubit Quantum Fourier transform as

$$\mathrm{QFT}\,|j_1 j_2 j_3\rangle = \frac{1}{\sqrt{8}}(|0\rangle + e^{2\pi i\ 0.j_3}\,|1\rangle) \otimes (|0\rangle + e^{2\pi i\ 0.j_2 j_3}\,|1\rangle) \otimes (|0\rangle + e^{2\pi i\ 0.j_1 j_2 j_3}\,|1\rangle). \tag{30}$$

We derive a more general product representation for the Quantum Fourier transform of n qubits.

Here j, k are integers $\in \{0, 1, ..., 2^n - 1\}$. We take digits for k

$$k = k_1 2^{n-1} + k_2 2^{n-2} + ... + k_n = \sum_{l=1}^{n} k_l 2^{n-l}.$$

$$\begin{aligned}
\mathrm{QFT}\,|j\rangle &= \frac{1}{2^{n/2}} \sum_{k=1}^{2^n} e^{2\pi i j k/2^n}\,|k\rangle \\
&= \frac{1}{2^{n/2}} \sum_{k_1=0}^{1}\sum_{k_2=0}^{1}\sum_{k_3=0}^{1} ... \sum_{k_n=0}^{1} e^{2\pi i j \sum_{l=1}^{n} 2^{n-l}/2^n}\,|k\rangle \\
&= \frac{1}{2^{n/2}} \sum_{k_1=0}^{1}\sum_{k_2=0}^{1}\sum_{k_3=0}^{1} ... \sum_{k_n=0}^{1} e^{2\pi i j \sum_{l=1}^{n} k_l 2^{-l}}\,|k_1 k_2 ... k_n\rangle \qquad \text{all digits of } k \\
&= \frac{1}{2^{n/2}} \sum_{k_1=0}^{1}\sum_{k_2=0}^{1}\sum_{k_3=0}^{1} ... \sum_{k_n=0}^{1} \bigotimes_{l=1}^{n} e^{2\pi i j k_l 2^{-l}}\,|k_l\rangle
\end{aligned}$$

and this is consistent with the above sum right above it.

We can move the sums for the digits inside the tensor product as each one only affects its own subspace

$$\begin{aligned}\mathrm{QFT}\,|j\rangle &= \frac{1}{2^{n/2}}\bigotimes_{l=1}^{n}\left[\sum_{k_l=0}^{1} e^{2\pi i j k_l 2^{-l}}\,|k_l\rangle\right] \\ &= \frac{1}{2^{n/2}}\bigotimes_{l=1}^{n}\left[|0\rangle + e^{2\pi i j 2^{-l}}\,|1\rangle\right]\end{aligned}$$

Now let's consider the digits of j. We take digits for j

$$j = j_1 2^{n-1} + j_2 2^{n-2} + ... + j_n = \sum_{m=1}^{n} j_m 2^{n-m}.$$

Consider the exponent

$$e^{2\pi i j 2^{-l}} = e^{2\pi i(\sum_{m=1}^{n} j_l 2^{n-m-l})}$$

if $n - m - l \geq 1$ then the exponent is a multiple of 2π and the factor is 1. This implies that each l value determines how many digits of j are important in the term. In terms of the digits of j and using fractional binary strings for j

$$\begin{aligned}\mathrm{QFT}\,|j\rangle = \frac{1}{2^{n/2}}&(|0\rangle + e^{2\pi i\ 0.j_n}\,|1\rangle) \otimes (|0\rangle + e^{2\pi i\ 0.j_{n-1}j_n}\,|1\rangle) \otimes (|0\rangle + e^{2\pi i\ 0.j_{n-2}j_{n-1}j_n}\,|1\rangle) \\ &\otimes ... \otimes (|0\rangle + e^{2\pi i\ 0.j_1 j_2 ... j_{n-1}j_n}\,|1\rangle).\end{aligned} \tag{31}$$

That this is consistent with the product representation for the 3 qubit quantum Fourier transform in equation 30. This product representation can make it easier to understand how to construct a quantum circuit to carry out the Quantum Fourier transform.

3.5 An efficient circuit for the Quantum Fourier Transform

The product representation of the Quantum Fourier Transform in Equation 31 can be turned into an efficient circuit shown in Figure 7. The circuit uses the phase gate

$$R_k = \begin{pmatrix} 1 & 0 \\ 1 & e^{2\pi i/2^k} \end{pmatrix} \tag{32}$$

We start with $|j_1 j_2 ... j_n\rangle$ as input. Then apply a Hadamard to the first bit

$$H \otimes I \otimes I ...\,|j_1 j_2 ... j_n\rangle = \frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i\ 0.j_1}\,|1\rangle)\,|j_2 ... j_n\rangle\,.$$

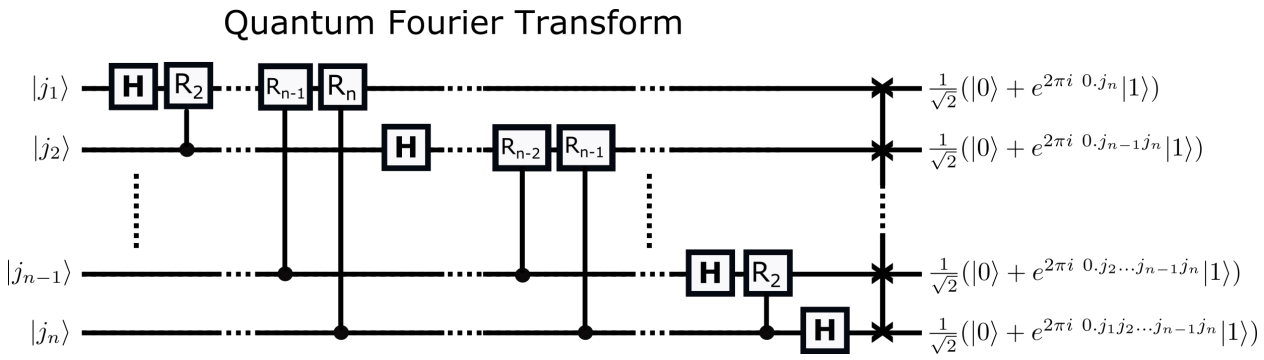

Figure 7: A circuit for the n qubit Quantum Fourier Transform. Here R_k is the phase gate shown in equation 32. The x's on the right are when all bits are reversed with swap operations.

This follows as $e^{2\pi i 0.j_1}$ is either 1 or -1 depending upon the value of the digit j_1. Now we apply a controlled R_2 phase gate with control bit the second bit and target bit the first one. The result is

$$\frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i\ 0.j_1j_2}\,|1\rangle)\,|j_2...j_n\rangle\,.$$

We continue operating with controlled phase gates. We do a controlled R_3 with control bit the third qubit and target the first qubit. We do a controlled R_4 with control bit the fourth qubit and target the first qubit, and so on until we have reached the n-th bit with R_n. The result is

$$\frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i\ 0.j_1j_2..j_n}\,|1\rangle)\,|j_2...j_n\rangle\,.$$

We then repeat this procedure (first with a Hadamard and then a series of phase gates) but for the second qubit applying controlled R_2 through R_{n-1} all with target bit the second bit. The result is

$$\frac{1}{2}(|0\rangle + e^{2\pi i\ 0.j_1j_2..j_n}\,|1\rangle) \otimes (|0\rangle + e^{2\pi i\ 0.j_2j_3..j_n}\,|j_3...j_n\rangle).$$

After repeating the procedure until the final qubit is reached with a Hadamard the final state is

$$\frac{1}{2^{n/2}}(|0\rangle + e^{2\pi i\ 0.j_1j_2..j_n}\,|1\rangle) \otimes (|0\rangle + e^{2\pi i\ 0.j_2..j_n}\,|1\rangle) \otimes (|0\rangle + e^{2\pi i\ 0.j_3..j_n}\,|1\rangle) \otimes ....$$
$$... \otimes (|0\rangle + e^{2\pi i\ 0.j_{n-1}j_n}\,|1\rangle) \otimes (|0\rangle + e^{2\pi i\ 0.j_n}\,|1\rangle).$$

Lastly we swap the order of all the bits giving as final state

$$\frac{1}{2^{n/2}} \otimes (|0\rangle + e^{2\pi i\ 0.j_n} |1\rangle) \otimes (|0\rangle + e^{2\pi i\ 0.j_{n-1}j_n} |1\rangle) \otimes ....$$
$$.... \otimes (|0\rangle + e^{2\pi i\ 0.j_3..j_n} |1\rangle) \otimes (|0\rangle + e^{2\pi i\ 0.j_2..j_n} |1\rangle) \otimes (|0\rangle + e^{2\pi i\ 0.j_1j_2..j_n} |1\rangle).$$

This is equivalent to Equation 31 so we have shown that the circuit shown in Figure 7 carries out an n-bit quantum Fourier transform.

A recursive way to describe the circuit that is related to the recursive FFT is explained in section 7.8 of the clear book by Rieffel & Polak.

4 Algorithms that use the Quantum Fourier Transform

Simon's algorithm does not use the Quantum Fourier transform but we discus it here because of its similarity to Shor's algorithm.

4.1 Simon's problem

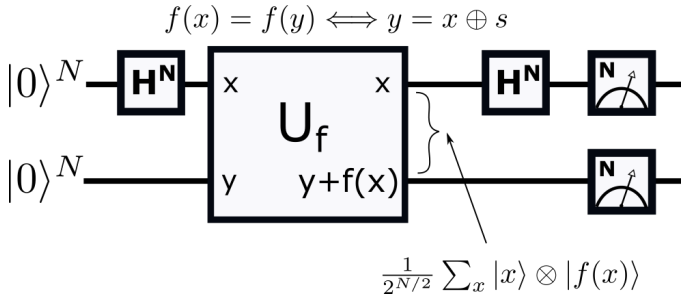

Figure 8: A circuit for the Simon's algorithm. The function satisfies $f(x) = f(y)$ iff $y = x \oplus s$ for a mystery N bit string s. The goal is to find the string s with a minimum number of queries of U_f. The circuit is run $O(N)$ times to determine s.

We once again are given an oracle function

$$f: \qquad \{0,1\}^N \to \{0,1\}^N$$

but it takes an N bit binary string to an N bit binary string. We assert that there is a secret N-bit string s (that is not all zeros) such that

$$f(x) = f(y) \Longleftrightarrow y = x \oplus s.$$

The relation is if and only if and the $\oplus$ is a bitwise XOR. If y has bits $y_1 y_2 ... y_N$, and likewise for x and s then each digit satisfies $y_i = x_i \oplus s_i$ for $i \in \{1, ...N\}$. For any x, y such that $y = x \oplus s$ then $f(x) = f(y)$ and vice versa.

Let's see how this works with 3 bits. Suppose the function gives

$$\begin{aligned} f(000) &= 110 \\ f(001) &= 010 \\ f(101) &= 110 \\ f(011) &= 001 \\ f(111) &= 101 \end{aligned}$$

The outputed values are not important. The important thing to notice is when two of the outputs agree, $f(000) = f(101)$. We can take the XOR of the inputs to find the mystery string s

$$s = 000 \oplus 101 = 101.$$

Efficiency: With Simon's problem, the goal is to find the secret string s by querying U_f as few times as possible. The problem is solved at **high probability** with $O(N)$ queries of U_f followed by $O(N^2)$ steps to solve some equations. The best that a classical algorithm can do is $O(2^{N/2})$ calls to $f()$. In algorithms we discussed previously (e.g., the Deutsch, Deutsch-Jozca and Bernstein-Vazirani problems) we solved the problem. Here we need to repeat the query to solve the problem at high probability. There are similarities to Simon's algorithm and the Shor algorithm approach to factoring. Simon's algorithm does not use the Quantum Fourier transform but we discus it here because of its similarity to Shor's algorithm.

For Simon's problem, the oracle is the unitary transformation

$$U_f : \qquad |x, y\rangle \rightarrow |x, y + f(x)\rangle \,. \tag{33}$$

We are working with two registers of qubits and each register is N bits long. The total number of qubits required for the calculation is $2N$ and the circuit we will use is shown in Figure 8.

Simon's algorithm is as follows: Start with

$$|000..\rangle \otimes |000..\rangle = |0\rangle^N \otimes |0\rangle^N \,.$$

Apply an N bit Hadamard to the first N bits. This gives

$$H^N \otimes I^N \, |000..\rangle \otimes |000..\rangle = \frac{1}{2^{N/2}} \sum_{x=0}^{2^N - 1} |x\rangle \otimes |000..\rangle$$

Apply U_f the oracle, giving

$$U_f(H^N \otimes I^N)\,|0\rangle^N \otimes |0\rangle^N = \frac{1}{2^{N/2}} \sum_{x=0}^{2^N-1} |x\rangle \otimes |f(x)\rangle\,. \tag{34}$$

We perform an N bit Hadamard again on the first N bits. This gives

$$|\psi\rangle_{\text{final}} = \frac{1}{2^N} \sum_{x=0}^{2^N-1} \sum_{y=0}^{2^N-1} (-1)^{x\cdot y}\,|y\rangle \otimes |f(x)\rangle \tag{35}$$

Then all bits are measured.

Suppose we measure z in the first register (the first N qubits) and w in the second register (the second N bits). There will be a set of x values that satisfy $f(x) = w$. The x values will come in pairs as if $f(x_a) = w$ then $x_b = x_a \oplus s$ satisfies $f(x_b) = w$. The amplitude of the relevant part of the wave vector

$$\begin{aligned} a_{z,w} &= \frac{1}{2^N} \sum_{x\in\{x_{1a},x_{1b},x_{2a},x_{2b},\ldots\}} (-1)^{x\cdot z}\,|z\rangle \otimes |w\rangle \\ &= \frac{1}{2^N} \sum_{\text{pairs}} \left((-1)^{x_{ia}\cdot z} + (-1)^{(x_{ia}\oplus s)\cdot z}\right) |z\rangle \otimes |w\rangle\,. \end{aligned}$$

For w to be measured this amplitude cannot be zero. There must be a pair of x values for which

$$(-1)^{x_{ia}\cdot z} = (-1)^{(x_{ia}\oplus s)\cdot z}.$$

This is true if

$$x_{ia}\cdot z = (x_{ia}\oplus s)\cdot z \qquad \text{mod } 2$$

$$\implies s\cdot z = 0 \qquad \text{mod } 2.$$

Each time you run the circuit you get a new value of z such that

$$z\cdot s = 0.$$

Once you have N different linearly independent measured z, values you can solve for s. To solve for s you need to run the algorithm $\mathrm{O}(N)$ times and then solve the $\mathrm{O}(N)$ equations which requires $\mathrm{O}(N^2)$ classical operations.

4.2 Outline of Shor Algorithm

The aim is to find prime factors of a given positive integer M efficiently. The Shor algorithm is a bounded probability polynomial-time quantum (BPQ) algorithm for factoring integers. Factoring of integers is an ingredient of many encryption algorithms.

Shor's algorithm has two ingredients:

- A reduction from factoring to period finding.
- A quantum algorithm using the Quantum Fourier transform for period-finding that uses only a constant ($O(1)$) number of queries to a function f, and a polynomial number of computational steps.

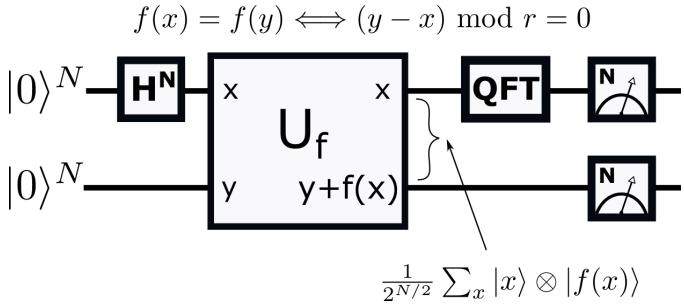

Figure 9: A quantum circuit for finding the period of a function. The function satisfies $f(x) = f(y)$ iff $y - x \bmod r = 0$ for a mystery integer r, known as the period. The period $0 < r < 2^N$. The goal is to find the period r with a minimum number of queries of U_f. The circuit is run order poly(N) times to determine the period r. Period finding used in Shor's factoring algorithm.

4.3 Period finding

Shor's algorithm for factoring uses a period finding algorithm.

Suppose we have a function f (an oracle) that takes one integer to another

$$f: \qquad \{0, ..., 2^{N-1}\} \to \{0, ..., 2^{N-1}\}.$$

We assert that r is a secret integer $0 < r < 2^N$ and

$$f(x) = f(y) \Longleftrightarrow y \bmod r = x \bmod r.$$

In other words $f(x) = f(x + mr)$ for all x, m. Here x, y, r, m are integers modulo 2^N and the arithmetic is also modulo 2^N. As the function $f()$ returns the same thing every r steps, the mystery integer r is the **period** of the function $f()$.

If you find an x_1 and an x_2 such that $f(x_1) = f(x_2)$ you are close to finding the mystery integer r. If $r \sim 2^{N/2}$ the oracle must be queried of order $2^{N/4}$ times to find a *collision* where two x values give the same $f(x)$ value. However, there is a quantum algorithm that computes r in order poly(N) queries of the oracle.

The beginning of the circuit (see Figure 9) is similar to that used in Simon's algorithm. We start with $|0\rangle^N \otimes |0\rangle^N$, perform an N bit Hadamard on the first register, then query the oracle. This gives the state

$$U_f(H \otimes I)\,|0\rangle^N \otimes |0\rangle^N = \sum_{x=0}^{2^N-1} \frac{1}{\sqrt{2^N}}\,|x\rangle\,|f(x)\rangle\,.$$

We perform a Quantum Fourier transform on the first register giving

$$|\psi\rangle_{\text{final}} = \sum_{x=0}^{2^N-1} \frac{1}{\sqrt{2^N}} \frac{1}{\sqrt{2^N}} \sum_{y=0}^{2^N-1} e^{2\pi i x y/2^N}\,|y\rangle\,|f(x)\rangle$$

We measure both registers. Suppose we measure z in the first register and w in the second register. Consider the set of x values that satisfies $f(x) = w$. We can find an x_w so that the set is $x_w, x_w + r, x_w + 2r, ...$ Suppose this set has N_w values in it. In other words N_w is the number of x values that satisfy $f(x) = w$. The relevant amplitude of the wave vector involves a sum over this set of possible x values,

$$\begin{aligned} a_{z,w} = \langle w|\,\langle z|\,|\psi\rangle_{\text{final}} &= \sum_{m=0}^{N_w-1} \frac{1}{2^N} e^{2\pi i (x_w + mr) z/2^N} \\ &= \frac{1}{2^N} e^{2\pi i x_w z/2^N} \sum_{m=0}^{N_w-1} e^{2\pi i m r z/2^N}. \end{aligned}$$

Let

$$q = e^{2\pi i r z/2^N}. \tag{36}$$

The amplitude $a_{z,w}$ involves the geometric series

$$\begin{aligned} \sum_{m=0}^{N_w-1} e^{2\pi i m r z/2^N} &= \sum_{m=0}^{N_w-1} q^m = 1 + q + ...q^{N_w-1} \\ &= \frac{1 - q^{N_w}}{1 - q}. \end{aligned}$$

The probability of measuring z

$$\text{Prob}(z) = \frac{1}{2^N} \left| \frac{1 - e^{2\pi i r z N_w / 2^N}}{1 - e^{2\pi i r z / 2^N}} \right|^2$$

If r is a power of 2 then q is a root of unity. The sum of the roots will be zero unless z is a multiple of $2^N/r$. A sequence of measurements for z will give multiples of $2^N/r$. The period r can be determined by find the greatest common divisor of this series of measurements.

When the period r does not divide 2^N , the transform approximates the exact case, so the amplitude is only high for integers z close to multiples of $2^N/r$. A sequence of measurements giving different z values can also be used to estimate the period r using a continued fraction expansion.

4.4 The Euclidean algorithm for finding the greatest common divisor of two natural numbers

Another ingredient needed in the Shor algorithm is the efficient algorithm known as the Euclidean algorithm for find the greatest common divisor of two positive integers.

The greatest common divisor of two natural numbers a, b we call $gcd(a, b)$.

Two integers are **relatively prime** or **coprime** if they share no prime factors. In other words if $gcd(a, b) = 1$ then a, b are relatively prime.

Here is the algorithm: Assume that $a > b$. Find q_0 and remainder r_0 such that

$$a = q_0 b + r_0.$$

Then find q_1 and remainder r_1 such that

$$b = q_1 r_0 + r_1.$$

Now find q_2 and remainder r_2 such that

$$r_0 = q_2 r_1 + r_2.$$

Now find q_3 and remainder r_3 such that

$$r_1 = q_3 r_2 + r_3$$

and so on.

A remainder r_N must eventually be equal to zero, at which point the algorithm stops. The final nonzero remainder r_{N-1} is the greatest common divisor of a and b.

The Euclidean algorithm on positive integers $a > b$ requires at most O(log a) steps,

4.5 Reduction of period finding to order finding

The input to the Shor algorithm is positive integer, M, the number to be factored.

The **order** of an integer a modulo M is the smallest integer $r > 0$ such that

$$a^r = 1 \bmod M.$$

If a, M are relatively prime then the order of a modulo M is infinite.

We can state that for a, M relatively prime $a^k \bmod M = a^{k+r} \bmod M$ if and only if $a^r = 1 \bmod M$.

Consider the function

$$f(k) = a^k \bmod M$$

for a, M relatively prime.

The function satisfies $f(k) = f(k+r)$ if and only if $a^r = 1 \bmod M$. As $f(k) = f(k+r)$ for all k, we say that $f()$ has period r.

In other words, for a relatively prime to M, the order r of a modulo M is the period of $f()$.

4.6 Finally the Shor algorithm

The relation between factoring and period finding inspires the algorithm. Here is the procedure to factor the integer M:

- Randomly choose a positive integer $a < M$. Use the Euclidean gcd algorithm to check whether a and M are relatively prime. If a and M are not relatively prime, then we have found a factor of M. If a and M are relatively prime, then proceed.
- Measure the period r of the function $f(k) = a^k \bmod M$.
- If r is even then

  $$a^r = 1 \bmod M$$

  can be written as

  $$a^{r/2} a^{r/2} - 1 = 0 \bmod M$$

  which can be written as

  $$(a^{r/2} - 1)(a^{r/2} + 1) = 0 \bmod M.$$

  Use the Euclidean algorithm to find the greatest common divisor of $(a^{r/2} + 1)$ and M. Use the Euclidean algorithm to find the greatest common divisor of $(a^{r/2} - 1)$ and M. If a greatest common divisor is nontrivial then we have found a factor M.
- Repeat the recipe!

Shor's algorithm is this recipe, but the period finding is done with the Quantum Fourier transform.

Apparently odd values of period r don't happen more than about 1/2 the time. The number of times you need to run the QFT depends on the number of positive integers less than r that are relatively prime to r (is this correct?) and this is given by the Euler ϕ function. As this function is logarithmically dependent on r, you don't need to run the QFT very many times to find the period.

This concludes our outline of the Shor algorithm.

Some holes could be filled in at later times: How many qubits do you need? How does the period finding work when r is not a power of 2? How many integers a do you need to chose to factor M at high probability? How efficient is the algorithm?

5 What is a Quantum Computer?

What are the requirements for a quantum computer?

- A set of quantum bits or few state quantum systems.
- Long coherence times in each quantum system. You need to be able to perform operations on the states before coherence is lost.
- The ability to perform some operations on individual qubits (or the individual few-state quantum systems).
- The ability to perform operations on pairs of qubits that entangle them.
- The ability to prepare quantum states.
- The ability to make measurements.

All qubits need not be localized to the same region. (Teleportation!)

Much of modern physics strives to be formulated in a geometric or coordinate free formalism. However in this set of notes, and as commonly done in introductions to quantum information theory, we started with a preferred basis. The preferred bases depends on how the entire system is decomposed into subsystems. For example we used basis vectors for a 2 qubit systems that look like $|01\rangle$ and $|00\rangle$. A 2 qubit system is equivalent to a 4 dimensional Hilbert space. Why do we do our calculations assuming a specific basis? Why not just work with generators of $U(4)$?

Realizations of quantum computers are probably going to be strongly tied to particular basis states. Feasible quantum gates are likely to operate in physically bounded regions in space. Likewise errors, decoherence and connections or interactions between qubits are going to be specifically implemented. Eventually more abstract versions of quantum computers might be realizable.

5.1 What is a Quantum Compiler?

A quantum algorithm is described by a single unitary matrix W. The goal of a quantum compiler is to find an optimal implementation of W using an elementary quantum gate set.

The meaning of *optimal* may depend on mechanisms for error correction in the presence of decoherence or noise and the number of available qubits.

Unitary transformations are reversible and no information is lost. However not all desired computations are reversible. With extra qubits you can always describe an irreversible function with a reversible one.

A NAND serves as a classical universal gate set, and the 3 qubit Toffoli gate, gives you a NAND operation but it takes an extra bit to do so. Each time a NAND is executed a qubit is wasted. Erasing bits costs energy (increases entropy) and is not reversible (or unitary). A way around this problem is to do the calculation, copy the output to a set of qubits, then reverse the computation, so that original used bits are reset. This saves the output and does not require any erasure. This procedure of computing and uncomputing saves space (needs fewer qubits) and maintains the entire calculation as a unitary operation, at the cost of using extra gates.

6 Quantum Error Correction

Currently quantum computers are limited because entangled states are fragile. If one of the qubit decoheres then the entanglement can be lost. If numerous gates are used, small errors may add up. A goal of a quantum **error correction** scheme is to use extra qubits to correct errors.

Classically, suppose instead of sending a 1 we send 111 and instead of sending 0 we send 000. If there is a single bit error, then the other two in the set of three can be used to detect and correct the error. An error in 2 or 3 bits of the three would be required for an error to remain undetected.

Are there analogies for quantum computing? On issue is that a qubit is not actually 1 or 0, rather it has a continuum of probabilities describing the likelihood that the system is in one of the states. In a superposition state of two states the relative phase between the two pieces is important.

Suppose you want the system to be $|0\rangle$ but actually the state is in $\sqrt{1-\epsilon^2}\,|0\rangle + \epsilon\,|1\rangle$. If you measure $|0\rangle$ you can ask, was there an error? If there was you no longer worry about it because the system is now in the $|0\rangle$ state. If you measure $|1\rangle$ you would know that there was an error and you would then call a Pauli-X operator to put the system back into $|0\rangle$.

Suppose we encode

$$\begin{array}{lll} |0\rangle & \text{as} & |000\rangle \\ |1\rangle & \text{as} & |111\rangle \end{array}$$

This code prevents bit flip errors but it would not necessarily prevent phase or sign errors. For example if $|000\rangle \to -|000\rangle$ we would not detect the error.

It turns out that, if a quantum error-correcting code protects against both bit-flip and sign/phase errors, then the code automatically protects against all possible 1-qubit errors.

We start with an entangled wave vector

$$|\psi\rangle_0 = \alpha\,|0\rangle\,|v\rangle + \beta\,|1\rangle\,|w\rangle\,.$$

A bit flip error on the first qubit gives

$$|\psi\rangle_1 = \alpha\,|1\rangle\,|v\rangle + \beta\,|0\rangle\,|w\rangle\,.$$

A sign or phase flip error on the first qubit gives

$$|\psi\rangle_2 = \alpha\,|0\rangle\,|v\rangle - \beta\,|1\rangle\,|w\rangle\,.$$

Note that $-1 = e^{\pi i}$ and can be thought of a phase. Both a bit flip error and a phase flip error on the first qubit gives

$$|\psi\rangle_3 = \alpha\,|1\rangle\,|v\rangle - \beta\,|0\rangle\,|w\rangle\,.$$

These four states represent a basis? for all possible states that can be reached by unitary transformation of the first qubit. (What if $\beta = 0$?, and the basis does not seem orthogonal?) This observation inspires the Shor 9-bit code for error correction.

6.1 Shor's 9-bit code

The proposal is to encode

$$\begin{aligned} |0\rangle \quad &\text{as} \quad \frac{1}{2^{3/2}}(|000\rangle + |111\rangle) \otimes (|000\rangle + |111\rangle) \otimes (|000\rangle + |111\rangle) \\ |1\rangle \quad &\text{as} \quad \frac{1}{2^{3/2}}(|000\rangle - |111\rangle) \otimes (|000\rangle - |111\rangle) \otimes (|000\rangle - |111\rangle). \end{aligned}$$

There are 9 states. You can think of the 9 states as a 3x3 matrix where each row protects against bit flip errors and each column protects agains phase flip errors.

Build a circuit that checks rows for bit flips and corrects them. Build a circuit that checks columns for phase flips and corrects them. After this is done the error correction procedure should correct errors not just in a wave vector that is $|0\rangle$ or $|1\rangle$ but also any superposition $\alpha\,|0\rangle + \beta\,|1\rangle$.

More general error correction codes use the CNOT, Hadamard and Phase gates (generating the Clifford group) to efficiently generate transpositions that can be used for error correction. Relevant is the Gottesman-Knill theorem which states that a quantum circuit constructed using only these gates can be simulated efficiently (in polynomial time) on a classical computer. I think that means it is efficient to do the error correction.