PHY265 Lecture notes: Introducing Quantum Algorithms

A. C. Quillen

April 16, 2024

Contents

1 Acronyms for Complexity and Terminology

We list some terminology.

1.1 Big O notation

Let $f, g : \mathbb{N} \to \mathbb{N}$ be functions of natural numbers. We say that $f(n) = O(g(n))$ iff $\exists\ \alpha, n_o \in \mathbb{N}$ such that $\forall n \geq n_o$ implies that $f(n) \leq \alpha g(n)$. The function $g(n)$ is an asymptotic upper bound for $f(n)$ and we say that f is of the order of g. Informally, $f(n) = O(g(n))$ means that f grows as g or slower.

1.2 P and NP and NP-complete and NP-hard

- A **decision problem** is a problem that gives a yes or no answer. The output is Boolean.
- **P**. The class of decision problems that can be solved in polynomial time. If the size of the input is N the decision problem can be solved on a deterministic sequential machine in an amount of time that is a polynomial function of the size of the input.
- **NP**. NP stands for 'nondeterministic polynomial time'. The class of decision problems for which an answer can be **verified** in polynomial time.
- **NP-complete**. If any NP problem can be transformed, in polynomial time, to a problem **A**, then the problem **A** is said to be NP complete. A problem **A** in **NP** is NP-complete if every problem in **NP** can be reduced to problem **A**. An NP-complete problem is an NP problem that is at least as "tough" as any other NP problem.
- **NP-hard**. Optimization problems are not decision problems. They often lack a function that lets you verify in polynomial time if you have an optimal solution so they are not in **P** or **NP**. However they can be computationally *as hard* as an **NP** problem if it takes an exponentially long time to find an optimal solution.

Can all problems that can be verified in polynomial time (**NP**) also be solved in polynomial time (in **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.

1.3 CIRCUT-SAT, 3-SAT, PSPACE, BPP, BQP

- **CIRCUIT-SAT**. You are given a Boolean circuit C that consists of a logical expression that is a function of Boolean variables, with each variable $x_i \in \{0, 1\}$. You want to find out if there is an $\mathbf{x}$ (a string of input bits) such that $C(\mathbf{x}) = 1$; and the output is TRUE. CIRCUIT-SAT $\in$ **NP** as $C()$, the circuit itself, is a verifying function that can be quickly computed. CIRCUIT-SAT is also **NP**-complete.
- **3-SAT**. A Boolean decision problem that can be written in terms of conjunctions of clauses, each of which only involve three Boolean variables. CIRCUIT-SAT can be reduced to K-SAT which can be reduced to 3-SAT. We discuss 3-SAT in more detail in section 6.4. 3-SAT is in the class of 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 or no 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**.

1.4 QFT, QPE, NISQ, VQA, VQE

- **QFT**. Quantum Fourier Transform.
- **QPE**. Quantum phase estimation.
- **NISQ**. Noisy Intermediate Scale Quantum. An acronym used to describe noisy quantum computers that have about 100 qubits.
- **VQC, VQA, VQE**. Variational Quantum Computer. Variational Quantum Algorithm. Variational Quantum Eigensolver. A hybrid algorithm that alternates between quantum algorithm, and measurement followed by classical computations to achieve optimization or minimization of a function of many variables.

1.5 Can quantum algorithms be faster than classical ones?

- **Nonexponential speedup.** Some quantum algorithms are demonstrably faster than the best classical algorithm, but not exponentially faster. These algorithms demonstrate that quantum algorithms are different than classical ones. Example: Grover's quantum search algorithm of an unsorted data base that achieves a quadratic speed-up compared to classical algorithms.

- **Exponential speed-up.** By feeding quantum superpositions to a quantum oracle, we can learn what is inside the oracle with an exponential speedup, compared to how long it would take if we were only allowed classical computations. An example is Simon's black box algorithm.

- **Exponential speedup for hard problems.** There are quantum algorithms that solve a problem in polynomial time, where the problem appears to be hard classically, so that it is strongly suspected that the problem is not in BPP. Example: Shor's factoring algorithm.

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

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

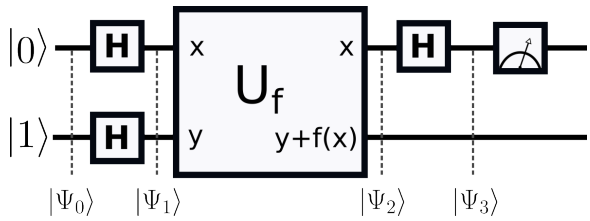

Figure 1: The Deutsch algorithm is the following quantum circuit applied to an initial state of $|01\rangle$. The U_f transformation is $|xy\rangle \to |x, y + f(x)\rangle$ where $x, y \in \{0, 1\}$ and the $+$ is mod 2. Here $f()$ is a Boolean function (returning 0 or 1). The goal is to determine if $f(x)$ 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.

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

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

and is only called a single time in the circuit.

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

$$|\Psi_0\rangle = |01\rangle$$

we apply $\mathbf{H} \otimes \mathbf{H}$

$$\begin{aligned} |\Psi_1\rangle &= \mathbf{H} \otimes \mathbf{H} \, |01\rangle \\ &= \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \\ &= \frac{1}{2}(|00\rangle - |01\rangle + |10\rangle - |11\rangle) \end{aligned}$$

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

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

The result is

$$|\Psi_2\rangle = \mathbf{U}_f \, |\Psi_1\rangle = \frac{1}{2}(|0, f(0)\rangle - |0, \text{NOT f}(0)\rangle + |1, f(1)\rangle - |1, \text{NOT } f(1)\rangle). \qquad (2)$$

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

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

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

$$\begin{aligned} |\Psi_3\rangle &= (\mathbf{H} \otimes \mathbf{I}) \, |\Psi_2\rangle \\ &= \frac{1}{\sqrt{8}}(|0, f(0)\rangle - |0, \text{NOT } f(0)\rangle + |0, f(1)\rangle - |0, \text{NOT } f(1)\rangle + \\ &\quad |1, f(0)\rangle - |1, \text{NOT } f(0)\rangle - |1, f(1)\rangle + |1, \text{NOT } f(1)\rangle \\ &= \frac{1}{\sqrt{8}}\Big(|0\rangle \otimes (|f(0)\rangle - |\text{NOT } f(0)\rangle + |f(1)\rangle - |\text{NOT } f(1)\rangle) \\ &\qquad |1\rangle \otimes (|f(0)\rangle - |\text{NOT } f(0)\rangle - |f(1)\rangle + |\text{NOT } f(1)\rangle)\Big) \end{aligned}$$

If $f()$ is a constant function then If $f(0)$ is equal to $f(1)$ and $|f(0)\rangle - |f(1)\rangle = 0$ and the final state evaluates to

$$|\Psi_3\rangle = \frac{1}{\sqrt{2}}(|0, f(0)\rangle - |0, \text{NOT } f(0)\rangle) \qquad \text{for } f() \text{ constant.}$$

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

If f is balanced, then $f(0)$ is opposite to $f(1)$. In this case $|f(0)\rangle - |NOTf(1)\rangle = 0$ and the final state is

$$|\Psi_3\rangle = \frac{1}{\sqrt{2}}(|1, f(0)\rangle - |1, \text{NOT } f(0)\rangle) \qquad \text{for } f() \text{ balanced.}$$

We notice that the first bit is now exclusively in the 1 state! Measurement of the first bit determines 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 query 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

$$\mathbf{W}_N \equiv \mathbf{H} \otimes \mathbf{H} \otimes \mathbf{H} \otimes ... \otimes \mathbf{H} = \mathbf{H}^{\otimes N}$$

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 $x_0x_1x_2$ and y has bits $y_0y_1y_2$

$$\begin{aligned} x \cdot y &= (x_0x_1x_2) \cdot (y_0y_1y_2) \\ &= (x_0 \wedge y_0) \oplus (y_1 \wedge y_1) \oplus (x_2 \wedge y_2) \\ &= x_0y_0 + y_1y_1 + x_2y_2. \end{aligned}$$

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

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

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

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

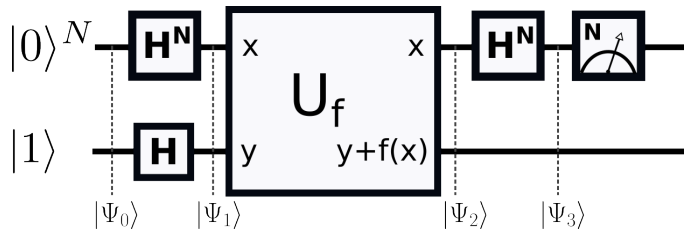

Figure 2: The Deutsch-Jozca algorithm is the following quantum circuit applied to an initial state of $|0\rangle^N \otimes |1\rangle$. Here the total number of qubits is $N+1$. The U_f transformation is $|xy\rangle \rightarrow |x, y+f(x)\rangle$ where $x \in (\{0,1)\})^N$, $y \in \{0,1\}$ and the $+$ 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.

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 initial state

$$|\Psi_0\rangle = |0\rangle^N |1\rangle .$$

The action of the circuit is

$$\begin{aligned}
|\Psi_1\rangle &= (H^{\otimes N} \otimes H) |0\rangle^N \otimes |1\rangle \\
&= \frac{1}{2^{N/2}} \sum_{x=0}^{2^N-1} |x\rangle \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \\
|\Psi_2\rangle &= U_f |\Psi_1\rangle \\
&= \frac{1}{2^{N/2+1/2}} \sum_{x=0}^{2^N-1} (|x\rangle \otimes |f(x)\rangle - |x\rangle \otimes |1+f(x)\rangle) \\
|\Psi_3\rangle &= (H^{\otimes N} \otimes I) |\Psi_2\rangle \qquad \text{using equation 3} \\
&= \frac{1}{2^{N+1/2}} \sum_{x=0}^{2^N-1} \sum_{y=0}^{2^N-1} (-1)^{x \cdot y} (|y\rangle \otimes |f(x)\rangle - |y\rangle \otimes |1+f(x)\rangle).
\end{aligned}$$

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

$$|\Psi_3\rangle = \frac{1}{2^{N+1/2}} \sum_{x=0}^{2^N-1} \sum_{y=0}^{2^N-1} (-1)^{x\cdot y}(-1)^{f(x)}(|y\rangle \otimes (|0\rangle - |1\rangle)) \tag{6}$$

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

$$|\Psi_3\rangle = \frac{1}{2^{N+1/2}} \sum_{x=0}^{2^N-1} \sum_{y=0}^{2^N-1} (-1)^{x\cdot y}(-1)^{c}(|y\rangle \otimes (|0\rangle - |1\rangle)) \qquad \text{for } f \text{ constant}$$

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

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

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.[1] Hence

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

The final wave function is

$$|\Psi_3\rangle = |0\rangle^N \otimes \frac{1}{\sqrt{2}}(-1)^{f(0)}(|0\rangle - |1\rangle) \quad \text{for } f() \text{ constant.}$$

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

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

$$|\Psi_3\rangle = \frac{1}{2^{N+1/2}} \sum_{x=0}^{2^N-1} \sum_{y=0}^{2^N-1} (-1)^{x\cdot y}(-1)^{f(x)}(|y\rangle \otimes (|0\rangle - |1\rangle)) \qquad \text{for } f() \text{ balanced.}$$

We need to compute the sum

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

[1] For example, consider $y = 1000$. When summing over all x values $x = 1000$ will give -1 in the sum and $x = 0000$ will give 1 in the sum. The two cancel out. Consider $x = 1ijk$ and $x = 0ijk$. These will cancel each other out for all possible values of ijk. Now consider $y = 1100$. For $x = 0000$ we get 1, for $x = 1000$ we get -1 for $x = 0100$ we get -1 and for $x = 1100$ we get 1. The 4 terms cancel. Likewise for any other possible values for the last two bits, the 4 terms cancel. The pattern should be clear.

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

$$\langle 0|^N |\Psi_3\rangle = \frac{1}{2^{N+1/2}} \sum_{x=0}^{2^N-1} (-1)^{f(x)}(|0\rangle - |1\rangle).$$

We need to compute

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

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

$$\langle 0|^N |\Psi_3\rangle = 0 \qquad \text{for } f() \text{ balanced.}$$

The state $|\Psi_3\rangle$ must be perpendicular to $|0\rangle^N$. This means that $|\Psi_3\rangle$ must contain a qubit that is not in the 0 state. 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.

Efficiency: For the Deutsch-Jozca black box problem 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 f. After a few queries (classically) 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

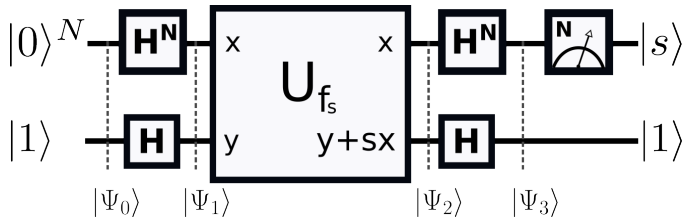

Figure 3: The Bernstein-Vazirani problem is solved with this quantum circuit applied to an initial state of $|0\rangle^N \otimes |1\rangle$. Here the total number of qubits is $N+1$. The U_f transformation is $|xy\rangle \to |x, y + f(x)\rangle$ where $x \in (\{0,1)\})^N$, $y \in \{0,1\}$ and the $+$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

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.

Our oracle does this

$$U_f \, |x, y\rangle = |x, y + s \cdot x\rangle \, .$$

Classically to find s you must query the function N times

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

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 $|0\rangle^N$, then apply the N-bit Hadamard operator $H^{\otimes N}$ then query with U_f, then apply the N-bit Hadamard operator again, the measure all bits. The result is s. The initial state

$$|\Psi_0\rangle = |0\rangle^N \otimes |1\rangle .$$

The action of the circuit is

$$\begin{aligned} |\Psi_1\rangle &= (H^{\otimes N} \otimes H) |0\rangle^N \otimes |1\rangle \\ &= \frac{1}{2^{N/2}} \sum_{x=0}^{2^N-1} |x\rangle \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \end{aligned}$$

$$\begin{aligned} |\Psi_2\rangle &= U_f |\Psi_1\rangle \\ &= \frac{1}{2^{N/2+1/2}} \sum_{x=0}^{2^N-1} (|x\rangle \otimes |f(x)\rangle - |x\rangle \otimes |1 + f(x)\rangle) \\ &= \frac{1}{2^{N/2+1/2}} \sum_{x=0}^{2^N-1} (|x\rangle \otimes |s \cdot x\rangle - |x\rangle \otimes |1 + s \cdot x\rangle) \\ &= \frac{1}{2^{N/2+1/2}} \sum_{x=0}^{2^N-1} (-1)^{s \cdot x} |x\rangle \otimes (|0\rangle - |1\rangle) \end{aligned}$$

as $s \cdot x$ is either 0 or 1.

$$(I \otimes H) |\Psi_2\rangle = \frac{1}{2^{N/2}} \sum_{x=0}^{2^N-1} (-1)^{s \cdot x} |x\rangle \otimes |1\rangle .$$

Lastly we use equation 3 to apply the last operation of $H^{\otimes N}$

$$\begin{aligned}
|\Psi_3\rangle &= (H^{\otimes N} \otimes H) |\Psi_2\rangle \\
&= \frac{1}{2^N} \sum_{x=0}^{2^N-1} \sum_{y=0}^{2^N-1} (-1)^{s\cdot x} (-1)^{x\cdot y} |y\rangle \otimes |1\rangle \\
&= \frac{1}{2^N} \sum_{x=0}^{2^N-1} \sum_{y=0}^{2^N-1} (-1)^{s\cdot (x+y)} |y\rangle \otimes |1\rangle . \qquad (8)
\end{aligned}$$

As we did previously (see equation 7) when computing the Deutsch Jozca algorithm, the sum

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

This follows because the number of positive and negative terms is the same and they cancel unless $q = 0$. Our sum in equation 8 is zero unless $s + y = 0$. Equation 8 becomes

$$|\Psi_3\rangle = \sum_{y=0}^{2^N-1} \delta_{s+y,0} |y\rangle \otimes |1\rangle .$$

Equivalently modulo 2, $y = s$ for the remaining term,

$$|\Psi_3\rangle = |s\rangle \otimes |1\rangle .$$

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!

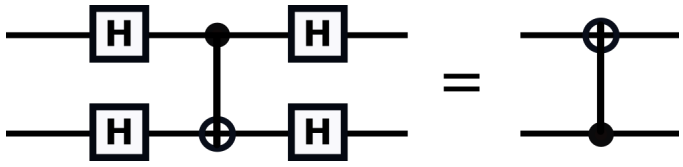

Figure 4: These two circuits are equivalent.

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.

Efficiency: Classically N queries of the function f are required to find the string s where N is the number of bits in s. In the quantum algorithm a single query sufficed to find s.

3 The Quantum Fourier Transform

The Quantum Fourier transform (QFT) is used in a number of 'fast' algorithms such as the Shor algorithm.

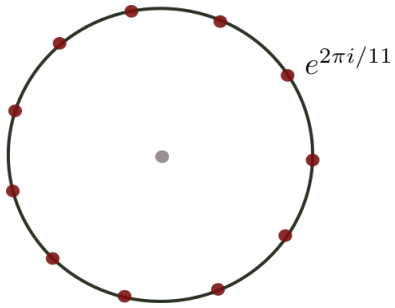

Figure 5: By symmetry, the roots of unity on the complex plane sum to zero.

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

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

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 $\sqrt{N}$. Sometimes 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

$$\begin{aligned} X_k &= \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j e^{\frac{2\pi i}{N}kj} \\ &= \frac{1}{N} \sum_{j=0}^{N-1} \sum_{l=0}^{N-1} X_l e^{-\frac{2\pi i}{N}lj} e^{\frac{2\pi i}{N}kj} \qquad \text{(using eqn 10)} \\ &= \frac{1}{N} \sum_{l=0}^{N-1} X_l \sum_{j=0}^{N-1} e^{\frac{2\pi i}{N}(k-l)j} \\ &= \frac{1}{N} \sum_{l=0}^{N-1} X_l \; N \; \delta_{lk} \qquad \text{(using eqn 11)} \\ &= X_k. \end{aligned}$$

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 unity (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 $\text{O}(N \log N)$.

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

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

Binary expansions		
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. It is computed modulo N.

Notice that equation 24 goes from 0 to $N-1$. With an n qubit system we take $N = 2^n$.

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

$$\mathrm{QFT}\,|x\rangle \equiv \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} e^{2\pi i x y/N}\,|y\rangle\,. \tag{25}$$

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.

$$\mathrm{QFT} \equiv \frac{1}{\sqrt{N}} \sum_{j,k=0}^{N-1} \omega_N^{jk}\,|j\rangle\,\langle k| \qquad \text{with} \qquad \omega_N = e^{2\pi i/N}. \tag{26}$$

The unitary matrix in equation 26 can also be regarded as a **definition for the Quantum Fourier transform**.

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 $N = 2^3 = 8$.

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

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

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

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

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

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

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

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

$$\begin{aligned}
\text{QFT}\,|j_1j_2j_3\rangle &= \frac{1}{\sqrt{8}}(|0\rangle + \omega^{4j_3}\,|1\rangle) \otimes (|0\rangle + \omega^{4j_2+2j_3}\,|1\rangle) \otimes (|0\rangle + \omega^{4j_1+2j_2+j_3}\,|1\rangle) \\
&= \frac{1}{\sqrt{8}}(|0\rangle + e^{2\pi i\frac{j_3}{2}}\,|1\rangle) \otimes (|0\rangle + e^{2\pi i\left(\frac{j_2}{2}+\frac{j_3}{4}\right)}\,|1\rangle) \otimes (|0\rangle + e^{2\pi i\left(\frac{j_1}{2}+\frac{j_2}{4}+\frac{j_3}{8}\right)}\,|1\rangle). \quad (27)
\end{aligned}$$

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

$$H = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix} \qquad P_{\frac{\pi}{2}} = \begin{pmatrix}1 & 0\\ 0 & i\end{pmatrix} \qquad P_{\frac{\pi}{4}} = \begin{pmatrix}1 & 0\\ 0 & e^{\frac{i\pi}{4}}\end{pmatrix}. \quad (28)$$

Equivalently

$$H = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & e^{\pi i}\end{pmatrix} \qquad P_{\frac{\pi}{2}} = \begin{pmatrix}1 & 0\\ 0 & e^{\frac{i\pi}{2}}\end{pmatrix} \qquad P_{\frac{\pi}{4}} = \begin{pmatrix}1 & 0\\ 0 & e^{\frac{i\pi}{4}}\end{pmatrix}. \quad (29)$$

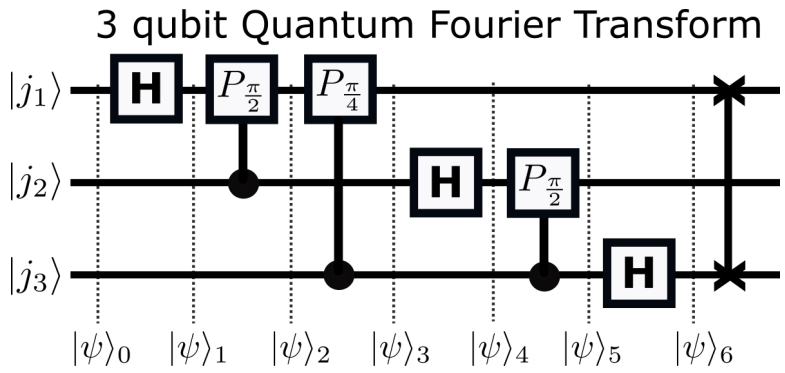

Figure 6: A circuit that computes the 3-qubit Quantum Fourier Transform. The 2-qubit gate on the right is a swap gate.

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 27.

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 27 which we repeat here

$$\mathrm{QFT}\,|j_1 j_2 j_3\rangle = \frac{1}{\sqrt{8}}(|0\rangle + e^{2\pi i \frac{j_3}{2}}\,|1\rangle) \otimes (|0\rangle + e^{2\pi i\left(\frac{j_2}{2}+\frac{j_3}{4}\right)}\,|1\rangle) \otimes (|0\rangle + e^{2\pi i\left(\frac{j_1}{2}+\frac{j_2}{4}+\frac{j_3}{8}\right)}\,|1\rangle). \tag{30}$$

Starting with $|\psi\rangle_0 = |j_1 j_2 j_3\rangle$ we perform a Hadamard operation on the first qubit

$$|\psi\rangle_1 = H \otimes I \otimes I\,|j_1 j_2 j_3\rangle = \frac{1}{\sqrt{2}}(|0\rangle + (-1)^{j_1}\,|1\rangle) \otimes |j_2, j_3\rangle$$

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

$$|\psi\rangle_2 = \frac{1}{\sqrt{2}}(|0\rangle + (-1)^{j_1} i^{j_2}\,|1\rangle) \otimes |j_2, j_3\rangle\,.$$

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

$$|\psi\rangle_3 = \frac{1}{\sqrt{2}}(|0\rangle + (-1)^{j_1} i^{j_2} e^{i\pi j_3/4}\,|1\rangle) \otimes |j_2, j_3\rangle\,.$$

This can also be written as

$$|\psi\rangle_3 = \frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i\left(\frac{j_1}{2}+\frac{j_2}{4}+\frac{j_3}{8}\right)}\,|1\rangle) \otimes |j_2, j_3\rangle\,.$$

and we recognize an expression similar to that of last qubit in equation 30. We perform a Hadamard operation on the second qubit, the wave vector becomes

$$|\psi\rangle_4 = \frac{1}{2}(|0\rangle + e^{2\pi i\left(\frac{j_1}{2}+\frac{j_2}{4}+\frac{j_3}{8}\right)}\,|1\rangle) \otimes (|0\rangle + (-1)^{j_2}\,|1\rangle) \otimes |j_3\rangle\,.$$

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

$$\begin{aligned}|\psi\rangle_5 &= \frac{1}{2}(|0\rangle + e^{2\pi i\left(\frac{j_1}{2}+\frac{j_2}{4}+\frac{j_3}{8}\right)}\,|1\rangle) \otimes (|0\rangle + (-1)^{j_2} i^{j_3}\,|1\rangle) \otimes |j_3\rangle \\ &= \frac{1}{2}(|0\rangle + e^{2\pi i\left(\frac{j_1}{2}+\frac{j_2}{4}+\frac{j_3}{8}\right)}\,|1\rangle) \otimes (|0\rangle + e^{2\pi i\left(\frac{j_2}{2}+\frac{j_3}{4}\right)}\,|1\rangle) \otimes |j_3\rangle\,.\end{aligned}$$

We perform a Hadamard operation on the third qubit giving

$$|\psi\rangle_6 = \frac{1}{\sqrt{8}}(|0\rangle + e^{2\pi i\left(\frac{j_1}{2}+\frac{j_2}{4}+\frac{j_3}{8}\right)}\,|1\rangle) \otimes (|0\rangle + e^{2\pi i\left(\frac{j_2}{2}+\frac{j_3}{4}\right)}\,|1\rangle) \otimes (|0\rangle + (-1)^{j_3}\,|1\rangle).$$

Lastly we swap the first and third qubits giving

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

and this matches the product form of the 3-qubit QFT in equation 30.

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 27 and 30)

$$\mathrm{QFT}\,|j_1 j_2 j_3\rangle = \frac{1}{\sqrt{8}}(|0\rangle + e^{2\pi i \frac{j_3}{2}}\,|1\rangle) \otimes (|0\rangle + e^{2\pi i \left(\frac{j_2}{2}+\frac{j_3}{4}\right)}\,|1\rangle) \otimes (|0\rangle + e^{2\pi i \left(\frac{j_1}{2}+\frac{j_2}{4}+\frac{j_3}{8}\right)}\,|1\rangle). \tag{31}$$

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

$$\mathrm{QFT}\,|j_1 j_2 j_3\rangle = \frac{1}{\sqrt{8}}(|0\rangle + e^{2\pi i\ 0.j_3}\,|1\rangle) \otimes (|0\rangle + e^{2\pi i\ 0.j_2 j_3}\,|1\rangle) \otimes (|0\rangle + e^{2\pi i\ 0.j_1 j_2 j_3}\,|1\rangle). \tag{32}$$

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

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

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

The tensor product in the bottom line is consistent with the sum in the exponential in the line right above it.

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

$$\mathrm{QFT}\,|j\rangle = \frac{1}{2^{n/2}} \bigotimes_{l=1}^{n} \left[\sum_{k_l=0}^{1} e^{2\pi i j k_l 2^{-l}} |k_l\rangle \right] \tag{33}$$

$$= \frac{1}{2^{n/2}} \bigotimes_{l=1}^{n} \left[|0\rangle + e^{2\pi i j 2^{-l}} |1\rangle \right] \tag{34}$$

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

$$e^{2\pi i j 2^{-l}} = e^{2\pi i (\sum_{m=1}^{n} j_l 2^{n-m-l})}. \tag{35}$$

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

$$\mathrm{QFT}\,|j\rangle = \frac{1}{2^{n/2}} (|0\rangle + e^{2\pi i\ 0.j_n} |1\rangle) \otimes (|0\rangle + e^{2\pi i\ 0.j_{n-1}j_n} |1\rangle) \otimes (|0\rangle + e^{2\pi i\ 0.j_{n-2}j_{n-1}j_n} |1\rangle)$$
$$\otimes ... \otimes (|0\rangle + e^{2\pi i\ 0.j_1 j_2 ... j_{n-1} j_n} |1\rangle). \tag{36}$$

That this is consistent with the product representation for the 3 qubit quantum Fourier transform in equation 32. 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 36 can be turned into an efficient circuit shown in Figure 7. The circuit uses the phase gate

$$R_k = \begin{pmatrix} 1 & 0 \\ 1 & e^{2\pi i/2^k} \end{pmatrix} \tag{37}$$

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

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

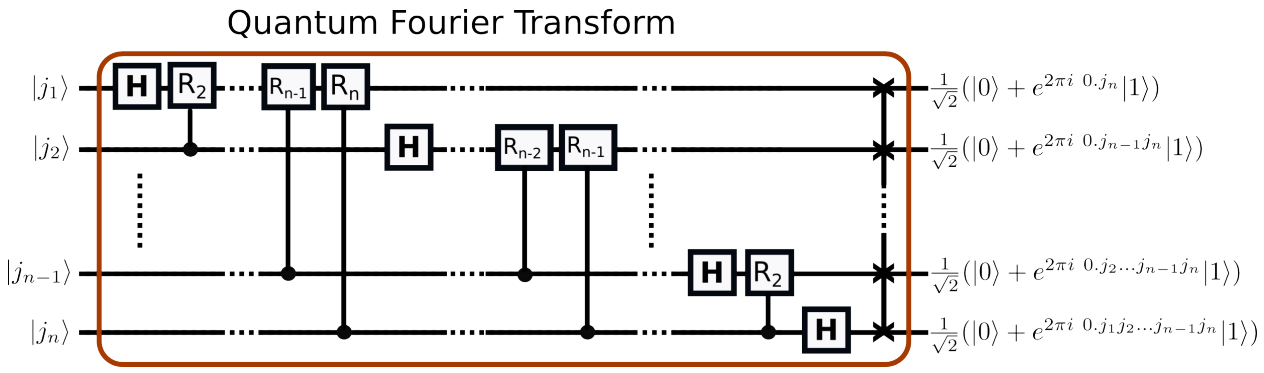

Figure 7: A circuit for the n qubit Quantum Fourier Transform. Here $R_k = \text{diag}(1, e^{2\pi i/2^k})$ is the phase gate shown in equation 37. The x's on the right are when all bits are reversed with swap operations.

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

$$\frac{1}{2^{n/2}} \otimes (|0\rangle + e^{2\pi i\ 0.j_n} |1\rangle) \otimes (|0\rangle + e^{2\pi i\ 0.j_{n-1}j_n} |1\rangle) \otimes .... \\ .... \otimes (|0\rangle + e^{2\pi i\ 0.j_3..j_n} |1\rangle) \otimes (|0\rangle + e^{2\pi i\ 0.j_2..j_n} |1\rangle) \otimes (|0\rangle + e^{2\pi i\ 0.j_1j_2..j_n} |1\rangle). \tag{38}$$

This is equivalent to Equation 36 so we have shown that the circuit shown in Figure 7 carries out an n-bit quantum Fourier transform.

3.6 Number of gates required for an n-bit QFT

How many gates are required to carry out a Quantum Fourier Transform of n qubits? Examination of Figure 7 gives $(n-1)(n-2)+1$ controlled phase gates + n Hadamard operators. This is $O(n^2)$ gates. The QFT is polynomial in n, the number of qubits, or equivalently polylog in N where $2^n = N$ is the number of quantum states.

3.7 Notes

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

It so happens that the unitary transformation for the Quantum Fourier transform satisfies

$$\mathbf{U}_{\mathrm{QFT}}^4 = \mathbf{I}.$$

We show that this is true: Starting with the definition for the QFT in equation 26

$$\begin{aligned}
\mathbf{U}_{\mathrm{QFT}} &= \frac{1}{\sqrt{N}} \sum_{jk} \omega^{jk} |j\rangle \langle k| \qquad \text{with} \quad \omega = e^{2\pi i/N} \\
\mathbf{U}_{\mathrm{QFT}}^2 &= \frac{1}{N} \sum_{jkl} \omega^{jk}\omega^{kl} |j\rangle \langle l| = \frac{1}{N} \sum_{jkl} \omega^{k(j-l)} |j\rangle \langle l| \\
&= \sum_{jl} \delta_{j-l} |j\rangle \langle l| \qquad \text{via equation 11} \\
&= \sum_{j} |j\rangle \langle -j| \\
\mathbf{U}_{\mathrm{QFT}}^4 &= \sum_{j} |j\rangle \langle j| = \mathbf{I}.
\end{aligned}$$

This implies that that the eigenvalues of $\mathbf{U}_{\mathrm{QFT}}$ are $\in \{1, -1, i, -i\}$. I don't think the eigenvectors are easy to compute.

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 period finding which is part of Shor's algorithm.

4.1 Simon's problem

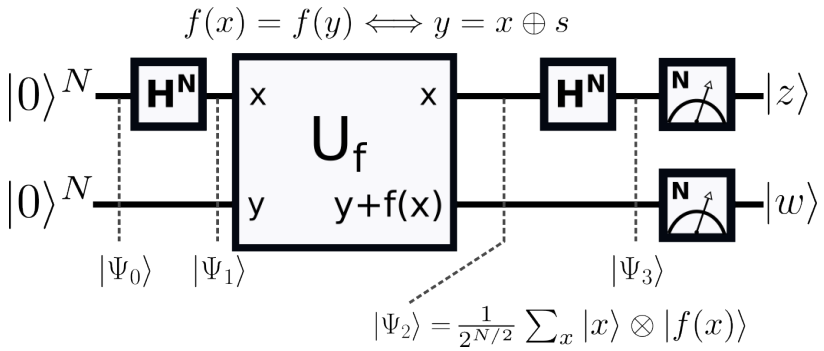

Figure 8: A circuit for the Simon's algorithm. The function satisfies $f(x) = f(y)$ iff $y = x \oplus s$ for a mystery N bit string s. The goal is to find the string s with a minimum number of queries of U_f. The circuit is run $O(N)$ times to determine s.

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

$$f(x) = f(y) \Longleftrightarrow y = x \oplus s. \tag{39}$$

Here x and y are also N bit strings. The relation is if and only if and the $\oplus$ is a bitwise XOR (an exclusive OR[2]). 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.

Consider digits $y_i = x_i \oplus s_i$.

What happens if I compute $y_i \oplus s_i = (x_i \oplus s_i) \oplus s_i$?

There are 4 cases to consider

[2]Gives a 0 if and only if each bit is the same.

x_i	s_i	$x_i \oplus s_i$	$(x_i \oplus s_i) \oplus s_i$
0	0	0	0
0	1	1	0
1	0	1	1
1	1	0	1

By considering the different cases we find $(x_i \oplus s_i) \oplus s_i = x_i$.

For string, this implies that if $y = x \oplus s$ then $y \oplus s = x$ and $(x \oplus s) \oplus s = x$. A solution x to $f(x) = w$ can be used to generate another solution $y = x \oplus s$ that satisfies $f(y) = w$.

Suppose we have $f(x) = f(y)$ which implies that $x = y \oplus s$. We can operate on both sides of the equation with $\oplus y$ to find $x \oplus y = y \oplus s \oplus y = s \oplus y \oplus y = s$. This means that if we find x, y such that $f(x) = f(y)$ we can find $s = x \oplus y$.

Suppose the function operates on 3 bits and gives

$$f(000) = 110$$
$$f(001) = 010$$
$$f(101) = 110$$
$$f(011) = 001$$
$$f(111) = 101$$

The outputed values are not important. The important thing to notice is when two of the outputs agree, $f(000) = f(101)$. We would say a *collision* has occurred. We can take the XOR of the two inputs from the collision to **identify** the mystery string s

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

4.1.1 Efficiency

With Simon's problem, the goal is to find the secret N-bit 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.

4.1.2 The algorithm - continued

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

$$U_f : \qquad |x, y\rangle \rightarrow |x, y + f(x)\rangle \,. \tag{40}$$

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

$$|\Psi_0\rangle = |000..\rangle \otimes |000..\rangle = |0\rangle^N \otimes |0\rangle^N .$$

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

$$|\Psi_1\rangle = H^{\otimes N} \otimes I^{\otimes N} |\Psi_0\rangle = \frac{1}{2^{N/2}} \sum_{x=0}^{2^N-1} |x\rangle \otimes |000..\rangle$$

Apply U_f the oracle, giving

$$|\Psi_2\rangle = U_f |\Psi_1\rangle = \frac{1}{2^{N/2}} \sum_{x=0}^{2^N-1} |x\rangle \otimes |f(x)\rangle . \tag{41}$$

We perform an N bit Hadamard again on the first N bits. Using equation 3, this gives

$$\begin{aligned} |\Psi_3\rangle &= H^{\otimes N} \otimes I^{\otimes N} |\Psi_2\rangle \\ &= \frac{1}{2^N} \sum_{x=0}^{2^N-1} \sum_{y=0}^{2^N-1} (-1)^{x\cdot y} |y\rangle \otimes |f(x)\rangle \qquad \text{before measurement.} \end{aligned} \tag{42}$$

Then all bits are measured. Here $x \cdot y$ refers to a bitwise multiply followed by a sum modulo 2.

Suppose we measure z in the first register (the first N qubits) and w in the second register (the second N bits). With z measured in the first register the sum in y collapses to a single term ($y \to z$). However, there could be a set of x values that satisfy $f(x) = w$. There must be at least one value of x that satisfies $f(x) = w$, otherwise we would not have been able to measure w. Let x_{1a} be a bit string that satisfies $f(x_{1a}) = w$. We compute $x_{1b} = x_{1a} \oplus s$. Because of the requirement on our function we know that $f(x_{1b}) = w$. Because $x \oplus s \oplus s = x$ we cannot generate other value of x that satisfies $f(x) = w$. However there could be another x_{2a} that satisfies $f(x_{2a}) = w$. From x_{2a} we can generate $x_{2b} = x_{2a} \oplus s$. Values of x that satisfy $f(x) = w$ must come in pairs.

The amplitude of the relevant part of the wave vector is

$$\begin{aligned} a_{w,z} = \langle z| \langle w|\Psi_3\rangle &= \frac{1}{2^N} \sum_{x\text{pairs}} (-1)^{x\cdot z} \\ &= \frac{1}{2^N} \sum_{\text{pairs}} \left((-1)^{x_{ia}\cdot z} + (-1)^{(x_{ia}\oplus s)\cdot z} \right) . \end{aligned}$$

For w to be measured this amplitude cannot be zero. There must be a pair of x values (that means there are two solutions) for which

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

This is true if

$$x_{ia}\cdot z = (x_{ia}\oplus s)\cdot z \qquad \text{mod } 2 \tag{43}$$

This expression is equivalent to[3] the condition

$$\implies s\cdot z = 0 \qquad \text{mod } 2. \tag{44}$$

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. Here N is the number of bits in the string s, equivalently the number of qubits. To solve for s you need to run the algorithm $O(N)$ times, giving $O(N)$ queries of the function, and then solve the $O(N)$ equations which requires $O(N^2)$ classical operations. Here N is the number of bits in the string s and is also the number of qubits.

4.2 Phase Estimation

Phase estimation is also an algorithm that can efficiently be implemented via the Quantum Fourier Transform. See section 6.2 below where we introduce the phase estimation algorithm and then use it as part of quantum searching algorithms. It is also used in the HHL algorithm for solving linear equations (section 7).

4.3 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:

[3] We use x_j, s_j, z_j to denote the j-th bits of the integers x_{ia}, s, z. Equation 43 can be written as $\sum_{j;z_j\neq 0} x_j = \sum_{j;z_j\neq 0} x_j \oplus s_j$ as only bits of z that are non-zero contribute. If $x_j = 0, s_j = 0$ then $x_j = x_j \oplus s_j = 0$ and terms on both side of equation 43 are zero. If $x_j = 1, s_j = 0$, then $x_j = x_j \oplus s_j = 1$ and terms on both side of equation 43 cancel. If $s_j = 0$ then it does not matter what x_j is. If $x_j = 0, s_j = 1$ then $x_j = 0$ but $x_j \oplus s_j = 1$ and equation 43 gains a 1 from the right hand side. If $x_j = 1, s_j = 1$ then $x_j = 1$ but $x_j \oplus s_j = 0$ and equation 43 gains a 1 from the left hand side. The number of bits with $s_j = 1$ (within the set $z_j = 1$) must be even for equation 43 to be satisfied. This means that only bits with $s_j = 1$ **and** $z_j = 1$ contribute to the sum and the number of bits that satisfy both conditions must be even. The condition that the sum is even means the sum is 0 mod 2. This gives equation 44.

- 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.

4.4 Period finding

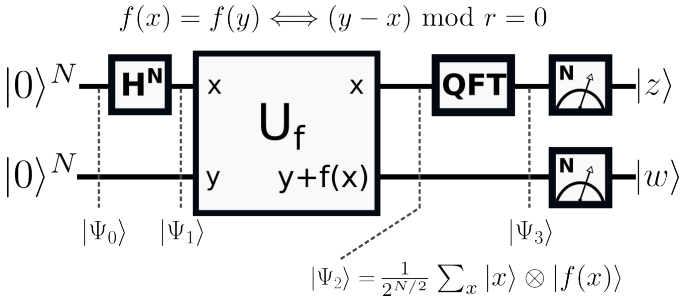

Figure 9: A quantum circuit for finding the period of a function. The function satisfies $f(x) = f(y)$ iff $y - x \bmod r = 0$ for a mystery integer r, known as the period. The period $0 < r < 2^N$. The goal is to find the period r with a minimum number of queries of U_f. The circuit is run order poly(N)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

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

$$f(x) = f(x + mr) \quad \text{for all} \quad x, m. \tag{45}$$

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 $r \sim 2^{N/2}$ and on a classical computer, 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

$$|\Psi_2\rangle = U_f(H \otimes I)\,|0\rangle^N \otimes |0\rangle^N = \sum_{x=0}^{2^N-1} \frac{1}{\sqrt{2^N}}\,|x\rangle\,|f(x)\rangle\,.$$

We perform a Quantum Fourier transform on the first register giving

$$|\Psi_3\rangle = \frac{1}{\sqrt{2^N}}\frac{1}{\sqrt{2^N}} \sum_{x=0}^{2^N-1}\sum_{y=0}^{2^N-1} e^{2\pi i x y/2^N}\,|y\rangle\,|f(x)\rangle\,. \tag{46}$$

We measure both registers. Suppose we measure z in the first register and w in the second register. The sum in y collapses to a single term $y \to z$. However, there can be multiple x values that satisfy $f(x) = w$. Consider the set of x values that satisfies $f(x) = w$. We can find an x_w so that the set is $x_w, x_w + r, x_w + 2r, ...$ Let N_w be the number of values in this set; in other words the number of possible x values that satisfy $f(x) = w$. The relevant amplitude of the wave vector involves a sum over this set of possible x values,

$$\begin{aligned} a_{z,w} = \langle w|\,\langle z|\Psi_3\rangle &= \sum_{m=0}^{N_w-1} \frac{1}{2^N} e^{2\pi i (x_w + mr) z/2^N} \\ &= \frac{1}{2^N} e^{2\pi i x_w z/2^N} \sum_{m=0}^{N_w-1} e^{2\pi i m r z/2^N}. \end{aligned}$$

Notice that this is independent of the actual value of w as x_w only contributes to a phase. Let

$$q = e^{2\pi i r z/2^N}. \tag{47}$$

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

$$\begin{aligned} \text{Prob}(z) &\propto |a_{w,z}|^2 \\ &\propto \left| \frac{1 - e^{2\pi i r z N_w/2^N}}{1 - e^{2\pi i r z/2^N}} \right|^2. \end{aligned}$$

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 $2^N/r$ and then in this case q is an integer. A sequence of measurements for z will give multiples of $2^N/r$. The period r can be determined by finding 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. Apparently the number of queries required is $O(\sqrt{N}) = O(\sqrt{\log m})$, where the integer r satisfies $0 < r < m = 2^N$, and $N = \log_2 m$ is the number of qubits. (This should be checked and explained!)

Here we have used N to be the number of qubits. However, we have previously used N to be the maximum size of an integer. We are going to try to be careful in each section about what N refers to!

The **continued fraction expansion** of a real positive number r is the following iterative algorithm which starts with $i = 0$:

1. Store $\lfloor r \rfloor$. This is the floor function and gives the greatest integer less than r. The stored number is an integer a_i where i refers to the iteration.

2. Compute $x = r - \lfloor r \rfloor$.

3. If $x = 0$ exit. Otherwise set $r = 1/x$, increment i and go to step 1.

The result is

$$r = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \ldots}}}.$$

The result is a series of non-negative integers $[a_0, a_1, ...]$.

4.5 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 finding 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. If $r_{N-1} = 1$ then a, b are **relatively prime**.

Here is an example with $a = 100, b = 15$,

$$\begin{aligned} 100 &= 6 \cdot 15 + 10 \\ 15 &= 1 \cdot 10 + 5 \qquad gcd \text{ is } 5 \\ 10 &= 2 \cdot 5 + 0. \end{aligned}$$

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

Below is an iterative form of the algorithm:

1. Set $r_{-1} = b$. Set $r_{-2} = a$. Set $i = 0$.
2. Compute $q_i = \left\lfloor \frac{r_{i-2}}{r_{i-1}} \right\rfloor$. (Again using the floor function).
3. Compute $r_i = r_{i-2} - q_i r_{i-1}$.
4. If $r_i = 1$ exit. In this case a, b are relatively prime.
5. If $r_i = 0$ exit. In this case the greatest common divisor of a, b is r_{i-1}.
6. Otherwise increment i and go to step 2.

4.6 Complexity of Factoring

Suppose our task is to factor an integer N with $n = \log N$ digits. A brute force algorithm goes through all primes p up to $\sqrt{N}$ and checks whether p divides N. The most efficient classical factoring algorithm, known as general number field sieve, achieves an asymptotic runtime that is exponential in $n^{\frac{1}{3}}$.

In contrast, Shor's factoring algorithm has runtime polynomial in n.

4.7 Reduction of period finding to order finding

Factoring can be related to another hard problem, that of finding the order of an integer which is equivalent to find a period.

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

Consider positive integers a, M that are **relatively prime**. The **order** of a modulo M is the *smallest* integer $r > 0$ such that

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

Furthermore[4]

$$a^k \bmod M = a^{k+r} \bmod M \qquad \text{if and only if} \qquad a^r = 1 \bmod M. \tag{48}$$

We show this by starting with

$$\begin{aligned} a^r &= 1 \mod M \\ a^r - 1 &= 0 \mod M \\ (a^r - 1)(a^k - 1) &= 0 \mod M \\ a^{r+k} - a^k - a^r + 1 &= 0 \mod M \\ a^{r+k} - a^k &= 0 \mod M. \end{aligned}$$

Going the opposite direction

$$\begin{aligned} a^{r+k} - a^k &= 0 \mod M \\ (a^r - 1)a^k &= 0 \mod M. \end{aligned}$$

Because a and M are relatively prime, a^k cannot be a multiple of M, so we can divide by a^k giving $a^r - 1 = 0 \mod M$.

Consider the function

$$f(k) \equiv a^k \bmod M \tag{49}$$

for a, M relatively prime, and where k is a positive integer. This 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. For a relatively prime to M, the order r of a modulo M is the period of the function $f()$.

Order-finding is believed to be a hard problem on a classical computer, in the sense that no algorithm is known to solve the problem using resources polynomial in the $O(n)$ bits needed to specify the problem, where $n = \log M$ is the number of bits needed to specify M.

[4]It is possible to show that a actually has an order $r \leq M$ by noting that the powers of a can only take a finite number of different values modulo M, and using the pigeonhole principle. Related is the fact that the powers of a modulo M (the integers $a^i \mod M$) form a group if a, M are relatively prime.

4.8 Finally the Shor algorithm – reducing factoring to period finding

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

- Step 1. Randomly choose a positive integer $a < M$. Use the Euclidean gcd (greatest common denominator) algorithm to check whether a and M are relatively prime. If a and M are not relatively prime, then their greatest common denominator is a factor of M! We have found a factor of M. If a and M are relatively prime, then proceed.
- Measure the period r of the function $f(k) = a^k \bmod M$. This is done using the quantum period finding algorithm.
- if r is odd then go back to Step 1.
- 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
$$(a^{r/2} - 1)(a^{r/2} + 1) = 0 \bmod M.$$
Note that $(a^{r/2} - 1)$ cannot be a multiple of M, otherwise the order of a would be $r/2$ and this contradicts the above statement that r is the order of a as $r > r/2$.
If $(a^{r/2} + 1)$ is a multiple of M then go back to Step 1.
- If $(a^{r/2}+1)$ is not a multiple of M then use the Euclidean algorithm to find the greatest common divisor of $(a^{r/2} - 1)$ and M. The greatest common divisor of $(a^{r/2} - 1)$ and M is a factor of M. We have found a factor of M.[5]

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 M that are relatively prime to M and this is given by the Euler ϕ function and is logarithmically dependent on $N = 2^n$. As this function is logarithmically dependent on N, you don't need to run the QFT very many times to find the period.

This concludes our outline of the Shor algorithm.

[5] gcd can't be 1 because the product of $(a^{r/2} + 1)(a^{r/2} - 1)$ is a multiple of M but neither are a multiple of M.

To implement Shor's algorithm on a quantum computer, it is necessary to construct unitary operations that perform modular multiplication and addition. The associated problem set illustrates some ways to achieve these operations. Modular arithmetic can be done with bitwise phase shifts and a Fourier transform. Multiplication can give a permutation of basis states (if the factor is relatively prime to the integer M for the mod).

Question: What happens if M is prime? Maybe you would notice that you only get odd values for r?

4.9 Factoring 35 with the Shor algorithm

In 2019 an attempt was made to factor the number 35 using Shor's algorithm on an IBM Q System One, but the algorithm failed. This is a good reason to work through the number 35 as an example.

Let $M = 35$. Suppose we choose integers a less than M without factoring M until we find one, that is relatively prime to M. On a quantum computer we would then devise a circuit that computes an oracle with $f(x)$ such that $f(x) = a^x \bmod M$ and then measure its periodicity.

Suppose we choose $a = 11$, relatively prime to M. We compute $a^x \bmod 35$ for $x = 1, 2, 3, 4..$ and find the sequence 1, 11, 16, 1, 11, 16 .... The period of this sequence is $r = 3$. The period is odd so we need to choose another value for a.

Suppose we choose $a = 13$, also relatively prime to 35. $a^x \bmod 35$ gives gives the sequence 1, 13, 29, 27, 1 ... The period of this sequence is $r = 4$. This is even! We compute $(a^{r/2} + 1) = a^2 + 1 = 170$ which is 30 mod 35. We verify that this is not a multiple of 35. The number $(a^{r/2} + 1)$ and M should have a common divisor, and they do, gcd(170,35) = 5. We have found a factor of M.

5 The hidden subgroup problem

Quantum algorithms that are known to be fast can be described as solving the following problem: Let $f()$ be a function from a finitely generated abelian group G to a finite set X such that f is constant on the cosets[6] of a subgroup H of G, and $f(g)$ is distinct on each coset; for $g, g' \in G$, $f(g) = f(g')$ iff $gH = g'H$. Equivalently the function satisfies

$$f(g) = f(g + h) \qquad \forall g \in G, h \in H \qquad \text{constant on cosets}$$

$$\text{for } g, g' \in G, \text{ if } g' \neq g + h \ \forall h \in H \text{ then } f(g) \neq f(g') \qquad \text{distinct on cosets.}$$

The goal is to find a generating set for the subgroup H.

[6] With $g \in G$, the left coset gH is the set of elements $\{gh : h \text{ an element of } H\}$. Here left cosets are the same as right cosets because the group is abelian.

- **Simon's problem.** Here G is the bitwise additive group $\mathbb{Z}_2^n = \{0,1\}^n$ of size $2n$, with operator $\oplus$ (bitwise addition modulo 2), the subgroup $H = \{0, s\}$ for a "hidden" n-bit Boolean string $s \in \{0,1\}^n$, and f satisfies $f(x) = f(y)$ iff $y = x \oplus s$. The function also returns an n-bit binary string. Finding the generator of H (i.e., finding s) solves Simon's problem.

- The **period finding** algorithm is also an example of the hidden subgroup problem. The group G is the set of natural numbers $\{0, ..., N-1\}$ which can be called $\mathbb{Z}_N$. The group operation is $+$ mod N. The function f operates on the set of natural numbers modulo N and is periodic with unknown (hidden) period r. The function obeys $f(x) = f(x+r)$. Assume that period r divides N (though this is not necessary for the algorithm, it does give a subgroup H in $\mathbb{Z}_N$). The function returns a number in the set $\mathbb{Z}_N$. Because f is periodic, it does not give N values, rather its range is a subset of unique possible values in the set of natural numbers $\{0, ..., N-1\}$.. The subgroup H is generated by r and has elements $\{0, r, 2r, 3r, ...\}$. The goal is to find the period r which determines the (hidden) subgroup.

- The **order finding** algorithm is also an example of the hidden subgroup problem. The group G is $\mathbb{Z}_M$ (natural numbers mod M) with the operation $+$ mod M. The function $f()$ operates on the set of natural numbers modulo M and depends on $a \in \mathbb{Z}_M$, via $f(k) = a^k$ mod M, where $0 \leq k \leq M$. The range of f is within the set $\mathbb{Z}_M$. The subgroup H depends on a hidden natural number r. The hidden r satisfies $a^r = 1$ mod M which implies that $f(k) = f(k+r)$. The subgroup H is generated by r and has elements $\{0, r, 2r, 3r, ...\}$. The goal is to find the period r which generates the (hidden) subgroup.

- The **discrete log** problem. We start with a generator $\gamma \in \mathbb{Z}_N$. This generates a cyclic group $\{\gamma^a | a \in \{0, ...N-1\}\}$ where numbers are modulo N. Given $A \in \mathbb{Z}_N$, the goal is to find a that satisfies $\gamma^a = A$. The number a is called the *discrete logarithm* of A with respect to the generator γ. Apparently, this problem is used in classical public-key cryptography.

  The group is $G = \mathbb{Z}_N \times \mathbb{Z}_N$ with operation $+$ mod N for two quantities. An element in G looks like (x, y). For example with $g_1 = (x_1, y_1)$ and $g_2 = (x_2, y_2)$, their sum is $g_1 + g_2 = (x_1 + x_2, y_1 + y_2)$ where both sums are modulo N. The generator γ and natural number A give a function $f : G \to X$ via

$$f(x, y) = \gamma^x A^{-y} \mod N. \tag{50}$$

  where the set X is within $\mathbb{Z}_N$. The subgroup H is generated by the group element $(a, 1)$ where a is hidden. The goal is to find a which generates the subgroup.

With $A = \gamma^a$

$$f(x_1, y_1) = \gamma_1^x A^{-y_1} = \gamma^{x_1 - a y_1}$$
$$f(x_2, y_2) = \gamma_2^x A^{-y_2} = \gamma^{x_2 - a y_2}.$$

$$\text{If } \; f(x_1, y_1) = f(x_2, y_2) \; \text{ then } \; (x_1 - x_2) = a(y_1 - y_2) \mod N. \tag{51}$$

What does the subgroup H generated by $(a, 1)$ look like? It contains elements

$$H : (x, y) \in \{(0,0), (a,1), (2a,2), (3a,3).....\}.$$

Note that these elements all satisfy $x = ay$ which (via equation 51) implies that

$$f(g) = f(g + h) \qquad \text{for } h \in H, g \in G.$$

Thus f is invariant on the cosets of H.

Hidden Subgroup Algorithms				
Algorithm	Simon's	Period finding	Order Finding	Discrete Log
Group G	$\{0,1\}^n$,$(+ \bmod 2)^n$	$\mathbb{Z}_N, + \bmod N$	$\mathbb{Z}_M, + \bmod M$	$\mathbb{Z}_N \times \mathbb{Z}_N, (+,+) \bmod N$
Subgroup H	$\{0, s\}, s \in \{0,1\}^n$	$\{0, r, 2r, ..\}$	$\{0, r, 2r, ..\}$	$\{(0,0), (a,1), (2a,2), ..\}$
Function f	$f(w) = f(w \oplus s)$	$f(x) = f(x+r)$	$f(k) = a^k \bmod M$ $f(k) = f(k+r)$	$f(x,y) = \gamma^x A^{-y} \bmod N$ $f(g) = f(g+h); h \in H, g \in G$
Set X	$\{0,1\}^n$	$\mathbb{Z}_N$	$\{a^j \bmod M\} \subset \mathbb{Z}_M$	$\{\gamma^j \bmod N\} \subset \mathbb{Z}_N$
Notes	G is isomorphic to $\mathbb{Z}_2^n$	r divides N for H a subgroup	choose $a \in \mathbb{Z}_M$ $a^r = 1 \bmod M$	choose $\gamma, A \in \mathbb{Z}_N$ $\gamma^a = A$
Goal	find bit-string s	find r	find r	find a

In the table w, s are bit-strings and $n, N, r, x, M, a, j, k, y, \gamma, A$ are natural numbers.

5.1 The discrete Fourier transform of a finite abelian group

For a nice introduction on the topic of Fourier transforms of abelian groups see Appendix B2 of the book by Rieffel and Polak.

We need a way to compute the quantum Fourier transform in the context of a group G. For finite abelian groups, a way to do this is to use the associated group of 1-dimensional representations of G. A d-dimensional **representation** of a group G is a map $\rho : g \to \rho(g)$ from G to the set of $d \times d$ invertible complex matrices, satisfying $\rho(gh) = \rho(g)\rho(h)$ for all $g, h \in G$ (giving a homomorphism). Here gh is the product of the two group elements with the group multiplication operator.

The **characteristic** $\chi_\rho(g)$ of a group element g is a complex number $\chi_\rho(g) = \mathrm{tr}(\rho(g))$ that is computed from the trace of the matrix representation of the group element.

Henceforth we assume that group G is finite and abelian and we consider only 1 dimensional representations; $d = 1$. This means that the matrices in the representation are just complex numbers. The characteristic functions and elements in the representation are the same functions; $\chi_\rho(g) = \rho(g)$. The characteristic of the group's identity element $\rho(e) = 1$ must be unity. The characteristics all must have modulus 1. All characteristics must be **roots of unity**. In other words for any non-identity element $g \in G$ that has order m (so $g^m = e$), its characteristic $\chi_\rho(g)$ should be in the form $\chi_\rho(g) = e^{2\pi ik/m}$ for some integer $0 < k < m$. This follows as $\chi_\rho(g^m) = [\chi_\rho(g)]^m = 1$.

The **fundamental theorem of finite abelian groups** states that every finite abelian group can be expressed as the direct sum of cyclic subgroups of prime-power order. In other words $G = \mathbb{Z}_{N_1} \otimes \mathbb{Z}_{N_2} .... \otimes \mathbb{Z}_{N_k}$ where N_i are the prime powers.[7] The total number of elements in G, known as the order of G, is $|G| = \prod_{i=1}^{k} N_i = N$. Any element in G can be written $g = g_1^{i_1} g_2^{i_2} ... g_k^{i_k}$ as a product of powers of the generators of the cyclic subgroups. For a 1d representation that means the characteristic of any element in the group can be written as a product of powers of the characteristics of the generators in each cyclic subgroup.

5.1.1 The group $\hat{G}$ of 1d representations

Consider two different 1d representations χ_i and χ_j of the finite abelian group G. We can make a third representation χ_k that satisfies $\chi_k(g) = \chi_i(g)\chi_j(g)$ for any $g \in G$. The identity representation is the trivial one $\chi_e(g) = 1$ for all $g \in G$. We can construct an inverse for any representation $[\chi_j]^{-1}$ via inverting all elements of another representation $[\chi_j]^{-1}(g) = [\chi_j(g)]^{-1}$. As characteristics are roots of unity $[\chi_j(g)]^{-1} = \chi_j^*(g)$ (the inverse is equal to the complex conjugate). The set $\{\chi_j\}$ of the different 1d representations of a finite abelian group G form a group that we denote $\hat{G}$.

We show an example of the group of representations for a cyclic group. The natural number $j \in \mathbb{Z}_N$ specifies a group element in the cyclic group $\mathbb{Z}_N$. Consider the function

$$\chi_k(j) = e^{2\pi ijk/N}. \tag{52}$$

Equation 52 resembles a discrete Fourier transform! The function $\chi_k()$ gives a complex number for each group element so specifies a particular representation. Each natural number k can generate a different 1d representation. Furthermore, $\chi_k(j + j') = \chi_k(j)\chi_k(j')$. The set $\{\chi_k\}$ for $k \in \{0, ..., N-1\}$ is the group of representations $\hat{G}$ for $\mathbb{Z}_N$. Notice that the number of elements in the group of representation is the same as the number of elements in the cyclic group; $|\hat{G}| = N$.

We describe a quantum system with N states with basis $\{|j\rangle\}$ with $j \in \mathbb{Z}_N$ which for

[7] Note that $8 = 2^3$ is an example of a prime to a power. Abelian groups of order 8 include $\mathbb{Z}_8$, $\mathbb{Z}_2 \otimes \mathbb{Z}_4$ and $\mathbb{Z}_2 \otimes \mathbb{Z}_2 \otimes \mathbb{Z}_2$.

the moment is also our group G. We define a state $|\chi_k\rangle$ as

$$|\chi_k\rangle \equiv \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} \chi_k(j) \, |j\rangle = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} e^{2\pi i j k/N} \, |j\rangle \, . \tag{53}$$

This is essentially a discrete Fourier transform. For every natural number $k \in \mathbb{Z}_N$ we can create a state $|\chi_k\rangle$. The transformation from basis set

$$\{|j\rangle\} \xrightarrow[\text{QFT}]{} \{|\chi_k\rangle\} \tag{54}$$

is given by the unitary matrix $U_{jk} = \frac{1}{\sqrt{N}} e^{2\pi i j k/N} \, |j\rangle \, \langle k|$ which gives the quantum Fourier transform (QFT) (see equation 25 and section 3.2).

For example, consider the cyclic group $\mathbb{Z}_4$. The group has four elements e, g, g^2, g^3. We could also refer to the elements as $0, 1, 2, 3$ with the operator + mod 4. We construct a particular 1d representation which we denote χ_a that has characteristics

$$\begin{aligned} \chi_a(e) &= 1 \\ \chi_a(g) &= e^{2\pi i/4} = e^{i\pi/2} = i \\ \chi_a(g^2) &= -1 \\ \chi_a(g^3) &= -i. \end{aligned}$$

All the other 1d representations for $\mathbb{Z}_4$ can be generated from powers of this representation. We label the presentations from their power of χ_a. This gives $\chi_1 = \chi_a$, $\chi_2 = \chi_a^2$, $\chi_3 = \chi_a^3$ and χ_0 the identity representation. The group of representations of $\mathbb{Z}_4$ is $\hat{G} = \{\chi_0, \chi_1, \chi_2, \chi_3\}$. We list the characteristics for each representation in the group of 1d representations.

Representations of $\mathbb{Z}_4$				
representation	$\chi(0)$	$\chi(1)$	$\chi(2)$	$\chi(3)$
χ_0	1	1	1	1
χ_1	1	i	-1	$-i$
χ_2	1	-1	1	-1
χ_3	1	$-i$	-1	i

The quantum Fourier transform gives basis elements

$$|\chi_k\rangle = \frac{1}{\sqrt{|G|}} \sum_{g \in G} \chi_a(g)^k \, |g\rangle \, .$$

The elements of the Fourier basis for $\mathbb{Z}_4$ are

$$|\chi_0\rangle = \frac{1}{2}(|0\rangle + |1\rangle + |2\rangle + |3\rangle)$$
$$|\chi_1\rangle = \frac{1}{2}(|0\rangle + i\,|1\rangle - |3\rangle - i\,|4\rangle)$$
$$|\chi_2\rangle = \frac{1}{2}(|0\rangle - |1\rangle + |3\rangle - |4\rangle)$$
$$|\chi_3\rangle = \frac{1}{2}(|0\rangle - i\,|1\rangle - |3\rangle + i\,|4\rangle).$$

Here each state is generated using a different representation in $\hat{G}$ and we have labelled them from the power of χ_a that was used to generate the representation.

More generally (meaning for a finite abelian group G that is more complicated than a cyclic group) we can similarly define a state associated with a specific 1d representation χ_k (in $\hat{G}$)

$$|\chi_k\rangle \equiv \frac{1}{\sqrt{|G|}} \sum_{g\in G} \chi_k(g)\,|g\rangle \tag{55}$$

where the sum is over all group elements. This equation essentially **defines the quantum Fourier transform for a finite abelian group**.

We give an example of the group of representations for a finite abelian group that is not cyclic. Consider the group $\mathbb{Z}_2 \otimes \mathbb{Z}_2$. This group has 4 elements which we call e, g, h, gh. Three of the elements have order 2 so the possible characteristic values are ± 1. The elements of the group could also be written as bit strings $\{00, 01, 10, 11\}$ with operator $\oplus$ or bit-wise addition, + mod 2. We list the characteristics for each representation in the group of 1d representations.

Representations of $\mathbb{Z}_2 \otimes \mathbb{Z}_2$				
representation	$\chi(00)$	$\chi(01)$	$\chi(10)$	$\chi(11)$
χ_0	1	1	1	1
χ_1	1	-1	1	-1
χ_2	1	1	-1	-1
χ_3	1	-1	-1	1

If a representation gives $\chi(e) = 1$, $\chi(g) = -1$, $\chi(h) = -1$ then we require that $\chi(gh) = \chi(g)\chi(h) = 1$.

Via equation 55 the Fourier basis for $\mathbb{Z}_2 \otimes \mathbb{Z}_2$ is

$$|\chi_0\rangle = \frac{1}{2}(|00\rangle + |01\rangle + |10\rangle + |11\rangle)$$
$$|\chi_1\rangle = \frac{1}{2}(|00\rangle - |01\rangle - |10\rangle + |11\rangle)$$
$$|\chi_2\rangle = \frac{1}{2}(|00\rangle - |01\rangle + |10\rangle - |11\rangle)$$
$$|\chi_3\rangle = \frac{1}{2}(|00\rangle + |01\rangle - |10\rangle - |11\rangle).$$

Some useful things that can be proven about the group of 1d representations of a finite abelian group:

- The number of elements in the group of 1d representations of a finite abelian group is the same as that in the original group;

  $$|\hat{G}| = |G|.$$

  To show this we use the fundamental theorem of finite abelian groups (the group can be written as a sum of cyclic groups). We construct the group of representations from the product of the group of representations for each cyclic component. We use the fact that the order of the group of representations for each cyclic group is equal to the order of that cyclic group. This implies that the number of representations for the full group is the product of the orders of the cyclic components which is equal to the order of the group. Take $G = \mathbb{Z}_{N_1} \otimes \mathbb{Z}_{N_2} .... \otimes \mathbb{Z}_{N_M}$ where N_i are prime powers. The total number of elements in G, is $|G| = \prod_{i=1}^{M} N_i = N$. Any element in G can be written $g = g_1^{i_1} g_2^{i_2} ... g_M^{i_M}$ with $i_j \in \mathbb{Z}_{N_j}$ for each index j. The quantum state corresponding to group element g is $|i_1 i_2 ... i_M\rangle$. The representations of the i-th cyclic group are the set $\{\chi_{i,k_j}\}$ with $k_j \in \mathbb{Z}_{N_i}$. The representations for the entire group are described by the product of the representations $\chi_{k_1,j_2,....k_M} = \chi_{1,k_1}\chi_{2,k_2}\cdots\chi_{M,k_M}$. The order of $\hat{G}$ is equal to the product of the orders of the representations for each cyclic group, which is equal to the order of the group. The construction illustrates that $|\hat{G}| = |G|$ for any finite abelian group.

- The Fourier states associated with two representations are **orthogonal**

  $$\langle \chi_k | \chi_j \rangle = \delta_{ij}. \tag{56}$$

  Here $\chi_k, \chi_j \in \hat{G}$ are two representations in the group of 1d representations of finite abelian group G. Here equality means the two representations are the same. Using

equation 55

$$\begin{aligned}
\langle\chi_k|\chi_j\rangle &= \frac{1}{|G|}\sum_{g',g\in G}\langle g'|\,\chi_k^*(g')\chi_j(g)\,|g\rangle = \frac{1}{|G|}\sum_{g\in G}\chi_k^*(g)\chi_j(g)\\
\chi_k(h)\,\langle\chi_k|\chi_j\rangle &= \frac{1}{|G|}\sum_{g\in G}\chi_k(h)\chi_k^*(g)\chi_j(g) \quad \text{for some } h\in G\\
&= \frac{1}{|G|}\sum_{g\in G}\chi_k^*(h^{-1}g)\chi_j(hh^{-1}g)\\
&= \frac{1}{|G|}\sum_{g'\in G}\chi_k^*(g')\chi_j(hg')\\
&= \chi_j(h)\frac{1}{|G|}\sum_{g\in G}\chi_k^*(g)\chi_j(g)\\
&= \chi_j(h)\frac{1}{|G|}\sum_{g\in G}\langle\chi_k|\chi_j\rangle\,.
\end{aligned}$$

If $\chi_j \neq \chi_k$ then for some $h \in G$, it must be that $\chi_j(h) \neq \chi_k(h)$. The above equation then implies that $\langle\chi_k|\chi_j\rangle = 0$ for $k \neq j$. If $j = k$ then because $\chi_j^*(g)\chi_j(g) = \chi_j(g^{-1}g) = 1$ for all g, the sum $\langle\chi_j|\chi_j\rangle = \sum_{g\in G}\frac{1}{|G|} = 1$. Taking into account both cases ($j = k$ and $j \neq k$) gives us the condition for orthogonality (equation 56).

Consider the complex vector space with $|G|$ elements. Because the Fourier states are orthogonal and because $|\hat{G}| = |G|$, the Fourier states span the entire complex vector space. The Fourier transform defined in equation 55 gives a change of basis $\{|g\rangle\} \to \{|\chi_k\rangle\}$. This means that the quantum Fourier transform for a finite abelian group is a unitary transformation. This implies that we can invert equation 55 giving the **inverse quantum Fourier transform for a finite abelian group**

$$|g\rangle = \frac{1}{\sqrt{|G|}}\sum_{k=0}^{|G|-1}[\chi_k(g)]^{-1}\,|\chi_k\rangle\,. \tag{57}$$

Notice that the sum is over all elements in the group of representations and we have used the fact that $|\hat{G}| = |G|$.

There is one more relation that will be useful when we again discuss the hidden subgroup problem. For a subgroup H of G and χ_k a 1d representation of G

$$\sum_{h\in H}\chi_k(h) = \begin{cases}|H| & \text{if } \chi_k(h) = 1, \forall h \in H\\ 0 & \text{otherwise.}\end{cases} \tag{58}$$

To show this we restrict χ_k to the subgroup, giving a 1d representation of H. Let χ_e be the identity representation of H giving $\chi_e(h) = 1, \forall h \in H$. Using equation 56 for 1d

representations of H

$$\delta_{ek} = \langle \chi_e | \chi_k \rangle = \frac{1}{|H|} \sum_{h \in H} \chi_k(h).$$

This implies that either χ_k is the identity representation and $\chi_k(h) = 1, \forall h \in H$ or the sum is zero. This statement then implies equation 58. Note that there is only one identity representation in $\hat{H}$ the group of representations of H. However there may be more than one representation in $\hat{G}$ the group of representations of G that acts like the identity representation when applied only to the elements of subgroup H.

It is helpful to **define the subgroup** $H^\perp \subset \hat{G}$ in the group of 1d representations of G that is related to the subgroup $H \subset G$:

$$H^\perp = \{\chi_k : \chi_k \in \hat{G} \text{ and } \chi_k(h) = 1 \ \forall h \in H\}. \tag{59}$$

The subgroup $H^\perp$ consists of 1d representations of G that are the identity representation when restricted to subgroup H. With this definition we update equation 58

$$\sum_{h \in H} \chi_k(h) = \begin{cases} |H| & \text{for } \chi_k \in H^\perp \\ 0 & \text{otherwise.} \end{cases} \tag{60}$$

Quantum Fourier transforms can be defined for all finite groups, however, for non-abelian groups the representation typically has higher dimension than $d = 1$. For a review see `https://arxiv.org/abs/0812.0380`.[8] Apparently, there are some cases of non-abelian groups where the Fourier transform can be computed efficiently.

5.2 The hidden subgroup finding algorithm

We use a tensor product vector space $\mathcal{H}_\mathcal{A} \otimes \mathcal{H}_\mathcal{B}$. The first space has dimension large enough to encompass the group G and the second space has dimension large enough to encompass the set X. We can use a basis for states

$$|g\rangle \, |x\rangle$$

based on group elements $g \in G$ and set elements $x \in X$. Recall that we should have a function $f : G \to X$. In other words $f(g)$ gives an element of the set X. The function f should satisfy $f(g + h) = f(g)$ for all $g \in G$ and $h \in H$ where the hidden subgroup is H. Here we are using $+$ to symbolize the group operation. Our goal is to find generators for the hidden subgroup.

We choose x_0, a member of the set X. We must construct a unitary operator (the oracle) that gives $U_f \, |g\rangle \, |x_0\rangle = |g\rangle \, |f(g)\rangle$. The oracle operator could be more generally

[8] Andrew M. Childs and Wim van Dam, Rev. Mod. Phys. 82, 1 (2010), *Quantum algorithms for algebraic problems*, `https://arxiv.org/abs/0812.0380`

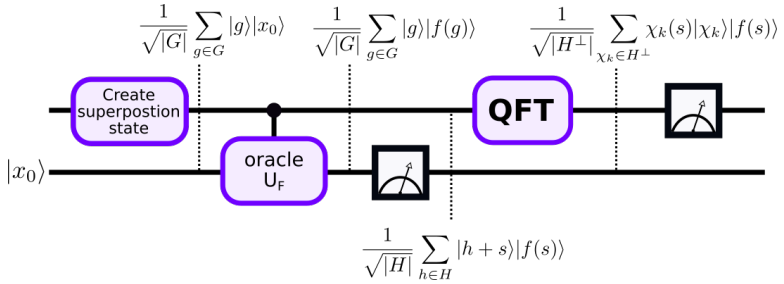

Figure 10: Steps of the hidden subgroup finding algorithm. We have finite abelian group G and hidden subgroup $H \subset G$. The oracle function $f : G \to X$ for a set X, satisfies $f(g+h) = f(g)$ for all $g \in G, h \in H$ and is distinct on different cosets. The top register is large enough to encompass elements of G. The bottom register is large enough to encompass elements of X. The quantum Fourier transform (QFT) is that generated by the group $\hat{G}$ of 1d representations of G (equation 55). Here $\chi_k : G \to \mathbb{C}$ refers to a 1d representation, giving a complex number that is a root of unity. The subgroup $H^\perp \subset \hat{G}$ is defined by equation 59. The goal is to find generators of H through measurements (in the last step) giving $\chi_k \in H^\perp$.

something like $U_f |g\rangle |x\rangle = |g\rangle |x + f(g)\rangle$ (this should look more familiar as it is used in Simon's algorithm and the period finding algorithm). Here the plus sign denotes a way to shift elements in X.

Here is the algorithm:

- Create a superposition state

$$|\psi\rangle = \frac{1}{\sqrt{|G|}} \sum_g |g\rangle |x_0\rangle .$$

- Operate on the superposition state with the oracle giving

$$U_f |\psi\rangle = \frac{1}{\sqrt{|G|}} \sum_g |g\rangle |f(g)\rangle .$$

- Measure the second register. This gives $f(s)$ for unknown $s \in G$. The first register collapses to a superposition over all $g \in G$ such that $f(g+s) = f(s)$. The values of g that satisfy this relation must be in the subgroup H. Again $+$ refers to the group operation. The state becomes

$$\frac{1}{\sqrt{|H|}} \sum_{h\in H} |h+s\rangle |f(s)\rangle . \tag{61}$$

- Apply a Fourier transform to the first register and then measure it!

  Taking the Fourier transform (using equation 55) of the first register in equation 61

$$\begin{aligned}
\mathrm{QFT}\frac{1}{\sqrt{|H|}}\sum_{h\in H}|h+s\rangle &= \frac{1}{\sqrt{|G||H|}}\sum_{k=0}^{|G|-1}\sum_{h\in H}\chi_k(h+s)\,|\chi_k\rangle \\
&= \frac{1}{\sqrt{|G||H|}}\sum_{k=0}^{|G|-1}\chi_k(s)\sum_{h\in H}\chi_k(h)\,|\chi_k\rangle \\
&= \frac{1}{\sqrt{|H^{\perp}|}}\sum_{\chi_k\in H^{\perp}}\chi_k(s)\,|\chi_k\rangle \qquad \text{using equation 58.} \qquad (62)
\end{aligned}$$

  What is measured in the last step of the algorithm is χ_k and this determines a particular 1d representation that must be in $H^{\perp}$. A few measurements should give quite a bit of information about the subgroup H. Note that the phases $\chi_k(s)$ do not affect the probabilities of these measurements.

The above steps can be repeated. With sufficient measurements in the last two steps, the generators of the subgroup can be efficiently recovered using properties of the measured characters.

To infer the generators of the subgroup H, one must look at the properties of $H^{\perp}$!

Why is this algorithm **efficient**? The sum in equation 61 only has only $|H|$ terms in it. The sum in equation 62 has only $|H^{\perp}|$ terms in it. Also Quantum Fourier transforms are fast. Quantum Fourier transforms can be computed in polylogarithmic time with respect to the size of the group G. With $|G| = N$ the number of qubits needed is $2^n = N$ so $\log_2 N = n$. Polylog means a polynomial of n. Previously we showed that the QFT could be done with $O(n^2)$ gates. State preparation can be done with an n-bit Hadamard which would be $O(n)$. The number operations that is required for the oracle (implementing the function f) would be polylog of N by leveraging Kitaev's theorem. Lastly one must count the number of times the algorithm needs to be run to ensure a high probability of finding the generators of the subgroup.

Currently efficient quantum algorithms are not known for every non-abelian hidden subgroup problem.

5.3 What does the subgroup $H^{\perp}$ look like?

- **Simon's problem.** The group G is $\{0,1\}^n$ with operator bitwise + mod 2. The group G is isomorphic to $\mathbb{Z}_2^n$. A 1d representation χ_r of G can be specified with a bit string $r = (r_1, r_2 ... r_n)$ with $r_i \in \{0,1\}$ giving $\chi_r(x) = (-1)^{r\cdot x}$ where $x = (x_1, x_2 ... x_n)$ is a bit string and $r \cdot x = \sum_i r_i x_i$ mod 2. Now consider $\chi_r(s) = (-1)^{r\cdot s}$ where bit string s specifies the hidden subgroup. The subgroup $H^{\perp}$ (as defined in equation 59)

consists of representations χ_r where $r \cdot s = 0$. Each measurement gives a bit string r. With enough measurements all bits of s can be determined.

- **Period finding.** The group G is cyclic as it is $\mathbb{Z}_N$. The representations χ_k are specified by integer k where $\chi_k(j) = \omega_N^{jk}$, j specifies the group element and $\omega_N = e^{2\pi i/N}$. The subgroup H consists of the set of elements $\{0, r, 2r...\}$ where natural number r is the hidden generator. Consider $\chi_k(rj) = \omega_N^{rjk}$ for natural number j. This must be equal to 1 for $\chi_k \in H^\perp$. The subgroup $H^\perp$ consists of representations χ_k where $rjk \bmod N = 0$. Each measurement gives a value for k. With k we can compute $r = \frac{mN}{jk}$ for unknown integers m, j. With a few measurements of k it should be possible to determine r.

- **Order finding.** Also a cyclic group! So probably equivalent to period finding!

- **Discrete Log.** The group is $\mathbb{Z}_N \otimes \mathbb{Z}_N$ with elements (x, y). The representations of this group can be described with two indices χ_{k_x,k_y}. The subgroup $H = \{(0,0), (a,1), (2a,2)...\}$. An element of the subgroup is (aj, j) with natural number j. $\chi_{k_x,k_y}(aj, j) = e^{2\pi i k_x ja/N} e^{2\pi i k_y j/N}$. For this to be equal to 1 (and that way $\chi_{k_x,k_y} \in H^\perp$) we require that $k_y j = 0 \bmod N$ and $k_x ja = 0 \bmod N$. The first condition gives $j = Nm/k_y$ for some integer m. Then the second condition gives $k_x Nm/k_y a = m'N$ or $a = \frac{m'}{m}\frac{k_y}{k_x}$ for two unknown integers m, m'. With a few measurements of possible values for k_x, k_y it should be possible to find a.

6 Quantum Search Algorithms

6.1 Grover's algorithm

Consider a search problem with M solutions out of $N = 2^n$ possibilities. Here n is the number of qubits. The oracle involves a Boolean function $f(x)$ which maps a set of N integers $\{0, 1, ..., N-1\} \to \{0, 1\}$. Here x a positive integer less than $N = 2^n$. The function returns 1 if a solution is found and 0 otherwise.

We assume that there are not many solutions. Consequently an algorithm for finding solutions is a search algorithm. The goal is to run a circuit $O(\sqrt{N/M})$ times, and then make measurements that would yield a solution to high probability.

A few unitary transformations are used in the algorithm.

- A black box transformation or oracle that implements the function $f()$.

- A conditional phase shift.

- The n bit Hadamard operator $W_n = H^{\otimes n}$.

The black box transformation, or oracle, is a unitary transformation that operates using the function $f()$

$$\begin{aligned} U_f \left|x, 0\right\rangle &\rightarrow \left|x, f(x)\right\rangle \\ U_f \left|x, 1\right\rangle &\rightarrow \left|x, f(x)+1\right\rangle \end{aligned} \tag{63}$$

With $x \in \{0, 1, ..., 2^n - 1\}$, the operator U_f operates on a 2^{n+1} qubit system. Here addition is modulo 2. The black box unitary transformation can also be written as

$$U_f \left|x, q\right\rangle \rightarrow \left|x, f(x)+q\right\rangle \tag{64}$$

with $q \in \{0, 1\}$.

What happens if the black box function operates on $|+\rangle$ or $|-\rangle$?

$$\begin{aligned} U_f \left|x+\right\rangle &= \frac{1}{\sqrt{2}}(\left|x, f(x)\right\rangle + \left|x, f(x)+1\right\rangle) \\ &= \left|x, +\right\rangle \\ U_f \left|x-\right\rangle &= \frac{1}{\sqrt{2}}(\left|x, f(x)\right\rangle - \left|x, f(x)+1\right\rangle) \\ &= \begin{cases} \left|x, -\right\rangle & \text{if} \quad f(x) = 0 \\ -\left|x, -\right\rangle & \text{if} \quad f(x) = 1 \end{cases} . \end{aligned} \tag{65}$$

We notice that if $f(x) = 1$ is a solution then the state vector picks up a minus sign. We also notice the $|x-\rangle$ is always an eigenstate of U_f. If x is a solution then the eigenstate has eigenvalue 1, otherwise it has eigenvalue -1. This means that we can describe the operation

$$U_f \left|x, -\right\rangle = (-1)^{f(x)} \left|x, -\right\rangle . \tag{66}$$

The oracle marks the solutions with a sign (equivalently a phase as $e^{\pi i} = -1$). Henceforth we are going to ignore the $|-\rangle$ work-bit as it remains unchanged during iterations of the algorithm.

We require an operator U_{PS} that performs a **conditional phase shift**. The operator $\times -1$ for all states except $|0\rangle^n$ which we can also write as $|0\rangle$ if we label basis states via integer base 2.

$$U_{PS}: \quad \left|x\right\rangle \rightarrow -(-1)^{\delta_{x0}} \left|x\right\rangle .$$

This operation should not affect additional work bits. The conditional phase shift can also be written as

$$U_{PS} = 2 \left|0\right\rangle \left\langle 0\right| - I \tag{67}$$

where $|0\rangle$ is the n-bit state $|0\rangle^{\otimes n}$ and I denotes the identity $I^{\otimes n}$.

Let's look at the operation of

$$U_{\text{IAM}} \equiv W_n U_{PS} W_n$$

on n bits. Here IAM means **Inversion About Mean**.

$$\begin{aligned} U_{\text{IAM}} &= W_n(2\left|0\right\rangle^n\left\langle 0\right|^n - I)W_n \\ &= 2W_n\left|0\right\rangle^n\left\langle 0\right|^n W_n - I. \end{aligned} \tag{68}$$

Recall that

$$W_n\left|0\right\rangle^n = \frac{1}{\sqrt{2^n}}\sum_{x=0}^{2^n-1}\left|x\right\rangle.$$

We define

$$\left|y\right\rangle = \frac{1}{\sqrt{2^n}}\sum_{x=0}^{2^n-1}\left|x\right\rangle. \tag{69}$$

This means that

$$W_n\left|0\right\rangle^n\left\langle 0\right|^n W_n = \left|y\right\rangle\left\langle y\right|. \tag{70}$$

Equation 68 becomes

$$U_{\text{IAM}} = 2\left|y\right\rangle\left\langle y\right| - I. \tag{71}$$

The operator U_{IAM} when applied to a state $\left|\psi\right\rangle = \sum_k \alpha_k\left|k\right\rangle$ produces

$$\begin{aligned} U_{\text{IAM}}\left|\psi\right\rangle &= (2\left|y\right\rangle\left\langle y\right| - I)\sum_k \alpha_k\left|k\right\rangle \\ &= \sum_{ijk}\frac{2}{N}\left|i\right\rangle\left\langle j\right|\alpha_k\left|k\right\rangle - \sum_k \alpha_k\left|k\right\rangle \\ &= \sum_i \left|i\right\rangle\frac{2}{N}\sum_j \alpha_j - \sum_k \alpha_k\left|k\right\rangle \\ &= \sum_k\left(\frac{2}{N}\sum_j \alpha_j - \alpha_k\right)\left|k\right\rangle \\ &= \sum_k (2\langle\alpha\rangle - \alpha_k)\left|k\right\rangle \end{aligned} \tag{72}$$

where the average

$$\langle\alpha\rangle = \frac{1}{N}\sum_k \alpha_k.$$

Because the operator U_{IAM} (as shown in equation 72) reverses the sign and adds the mean value, it **inverts about the mean** value.

It is helpful to construct two states that are based on eigenstates of the oracle transformation U_f. Equation 65 shows that $|x\rangle$ is always an eigenstate of U_f (assuming that the extra work-bit remains at $|-\rangle$) but with eigenvalue 1 if x is a solution, and with eigenvalue -1 otherwise. We construct a state that is a sum of the M states that are solutions and have $f(x) = 1$

$$|s\rangle = \frac{1}{\sqrt{M}} \sum_{x:f(x)=1} |x\rangle .$$

We construct a state that is a sum of the $N - M$ states that have $f(x) = 0$ and are not solutions

$$|v\rangle = \frac{1}{\sqrt{N-M}} \sum_{x:f(x)=0} |x\rangle .$$

These two states are orthogonal; $|v\rangle \langle s| = 0$. (This follows because they are comprised of separate sets of orthogonal states). Our mixed state $|y\rangle$ defined in equation 69 can be described in terms of these two states; $|s\rangle$ (where s is for solution) and $|v\rangle$ (where v is for vanilla).

$$\begin{aligned} |y\rangle &= \sqrt{\frac{N-M}{N}} |v\rangle + \sqrt{\frac{M}{N}} |s\rangle \\ &= \cos(\theta/2) |v\rangle + \sin(\theta/2) |s\rangle , \end{aligned} \tag{73}$$

where we define an angle θ that satisfies

$$\cos\left(\frac{\theta}{2}\right) \equiv \sqrt{\frac{N-M}{N}}, \qquad \sin\left(\frac{\theta}{2}\right) = \sqrt{\frac{M}{N}}. \tag{74}$$

We expect that θ is small if there are few solutions.

We compute the identity in the $|v\rangle, |s\rangle$ basis.

$$I = |v\rangle \langle v| + |s\rangle \langle s| . \tag{75}$$

With equation 73 we find

$$\begin{aligned} |y\rangle \langle y| =& \cos^2(\theta/2) |v\rangle \langle v| + \sin^2(\theta/2) |s\rangle \langle s| + \cos(\theta/2)\sin(\theta/2)(|v\rangle \langle s| + |s\rangle \langle v|) \\ =& \frac{(1+\cos\theta)}{2} |v\rangle \langle v| + \frac{(1-\cos\theta)}{2} |s\rangle \langle s| + \frac{\sin\theta}{2}(|v\rangle \langle s| + |s\rangle \langle v|), \end{aligned} \tag{76}$$

where I have used identities $2\cos^2 x - 1 = \cos(2x)$, $2\sin^2 x - 1 = -\cos(2x)$, $2\sin x \cos x = \sin(2x)$. We compute U_{IAM}, the operator that inverts about the mean in the $|v\rangle, |s\rangle$ basis. Using equations 71 and 75

$$\begin{aligned} U_{\mathrm{IAM}} &= 2 |y\rangle \langle y| - I \\ &= \cos\theta(|v\rangle \langle v| - |s\rangle \langle s|) + \sin\theta(|v\rangle \langle s| + |s\rangle \langle v|). \end{aligned} \tag{77}$$

In the $|v\rangle, |s\rangle$ basis

$$U_{\text{IAM}} = \begin{pmatrix} \cos\theta & \sin\theta \\ \sin\theta & -\cos\theta \end{pmatrix}. \tag{78}$$

The Grover algorithm is illustrated in Figure 12. It depends on an iteration step which is illustrated in Figure 11.

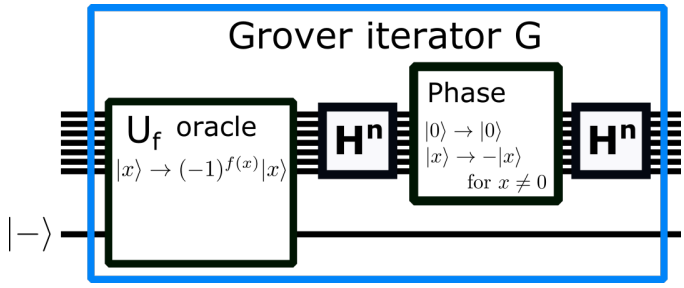

Figure 11: The Grover iterator that is part of the Grover search algorithm shown in Figure 12.

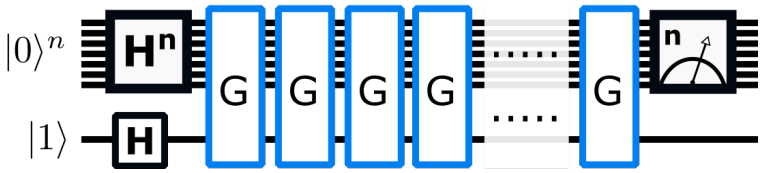

Figure 12: A quantum circuit for doing a quantum search with Grover's algorithm. The operator G is a Grover iterator shown in Figure 11.

The Grover iteration G (see Figure 11) is the following set of unitary operations:

- Apply the oracle function U_f.
- Apply $W_n \otimes I$.
- Perform the conditional phase shift, $U_{PS} \otimes I$.
- Apply $W_n \otimes I$.

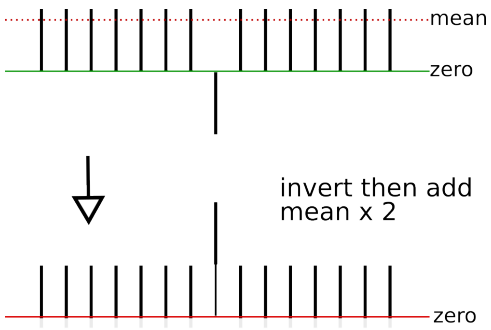

Figure 13: The Grover iteration inverts about the mean which amplifies the desired solutions.

An additional work bit remains at $|-\rangle$ througout.

Altogether the Grover iteration operator is

$$G = \left[(W_n U_{PS} W_n) \otimes I\right] U_f = (U_{IAM} \otimes I) U_f. \tag{79}$$

For the full algorithm (see Figure 12), the circuit starts with initial initial state $|y, -\rangle$ and then the Grover iterator G is applied a number of times. Finally n bits are measured.

Let's look at what happens during a single iteration.

The oracle function sends $|v\rangle \to |v\rangle$ but flips the sign of $|s\rangle$. Assuming that the work bit is set to $|-\rangle$, and we start with a state $a|v\rangle + b|s\rangle$, the oracle function does

$$U_f(a|v\rangle + b|s\rangle) \otimes |-\rangle = (a|v\rangle - b|s\rangle) \otimes |-\rangle .$$

In matrix form and in the $|v\rangle, |s\rangle$ basis (and ignoring the work bit)

$$U_f = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}. \tag{80}$$

Using the matrix form for U_{IAM} in the $|v\rangle, |s\rangle$ basis (shown in equation 78) the Grover iterator

$$G = \begin{pmatrix} \cos\theta & \sin\theta \\ \sin\theta & -\cos\theta \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{81}$$

$$= \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}. \tag{82}$$

This looks like a rotation by angle θ!

The algorithm starts with initial state $|y\rangle = \cos(\theta/2)\,|v\rangle + \sin(\theta/2)\,|s\rangle$. If we apply the Grover iterator to this state a single time then it is as if we have rotated the $|v\rangle$ state by an angle $3\theta/2$. In other words

$$G\,|y\rangle = \cos(3\theta/2)\,|v\rangle + \sin(\theta/2)\,|s\rangle\,.$$

After k applications of the Grover iterator

$$G^k\,|y\rangle = \cos\left(\frac{(2k+1)\theta}{2}\right)|v\rangle + \sin\left(\frac{(2k+1)\theta}{2}\right)|s\rangle\,. \tag{83}$$

If we apply G sufficient number of times that

$$\frac{(2k+1)\theta}{2} \sim \frac{\pi}{2} \tag{84}$$

then the state will only consist of possible solutions of $f()$. Here k is the number of times we need to apply the Grover iterator.

6.1.1 Efficiency

We defined θ with

$$\sin(\theta/2) = \sqrt{\frac{M}{N}}$$

(via equation 74). Assuming there are few solutions $\theta/2 \sim \sqrt{M/N}$. How many queries (k iterations of G) would we need? We insert $\theta \sim 2\sqrt{M/N}$ into equation 84 to find

$$4k\sqrt{\frac{M}{N}} \sim \pi$$

$$k \sim \frac{\pi}{4}\sqrt{\frac{N}{M}}.$$

The number of queries k of the function that are needed to find a solution is $O(\sqrt{N})$. Here N is the number of possible input strings ($N = 2^n$ where n is the number of qubits).

A classical search would require N/M queries (and so is $O(N)$) to find a solution, so the quantum algorithm is faster than a classical search, but it is not exponentially faster.

The final measurement doesn't always give you a correct answer! However it is unlikely you would get an incorrect answer. You could check a solution if you have access to the function f. Alternatively you could run the algorithm multiple times to make sure you get the same set of solutions.

If you don't know the number of solutions, it is difficult to choose the optical number of iterations k. This issue is addressed in the next section!

Apparently Grover's algorithm is essentially optimal (Bennet+97)[9] and it is impossible to do better than $O(\sqrt{N})$. For a nice explanation of the optimal aspect and a nice introduction to modifications of the Grover algorithm, see chap 9 of the lovely book by Rieffel and Polak.

6.2 Quantum Phase Estimation

Phases can be estimated using a quantum Fourier transform, so we could have discussed phase estimation in one of the previous sections. We discuss it here because equation 83 is sinusoidal and phase estimation is used in the quantum counting algorithm that we will discuss below.

We have a unitary transformation $\mathbf{U}$ and an eigenvalue of $\mathbf{U}$ that is $e^{2\pi i\phi}$ that depends on phase $\phi \in [0, 1)$. The associated eigenvector is $|u\rangle$.

We apply the circuit shown in Figure 14. Notice that after each application of $\mathbf{U}$, the bottom set of qubits remains in the state $|u\rangle$ because $|u\rangle$ is an eigenvector of $\mathbf{U}$.

We use a set of t qubits on the top part of the circuit, and each is affected by a different number of iterations of $\mathbf{U}$. The number of iterations of $\mathbf{U}$ applied in each step is a power of two. After the circuit in Figure 14 is implemented, the state is

$$|\psi\rangle = \frac{1}{2^{t/2}}(|0\rangle + e^{2\pi i(2^{t-1}\phi)}\,|1\rangle) \otimes (|0\rangle + e^{2\pi i(2^{t-2}\phi)}\,|1\rangle) \otimes ... \\ .... \otimes (|0\rangle + e^{2\pi i(2^{1}\phi)}\,|1\rangle) \otimes (|0\rangle + e^{2\pi i(2^{0}\phi)}\,|1\rangle) \otimes |u\rangle\,. \tag{85}$$

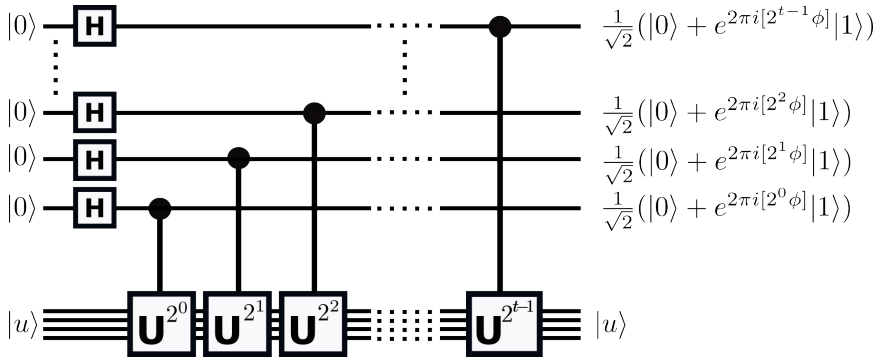

Figure 14: The beginning of a circuit to measure a phase ϕ. Here $|u\rangle$ is an eigenvector of unitary matrix $\mathbf{U}$ and $|u\rangle$ has eigenvalue $e^{2\pi i\phi}$. There are t qubits in the top register. For each qubit in the top register a controlled operation that is $\mathbf{U}$ unitary matrix to a power of 2 has target the bottom register.

[9] Bennett C.H., Bernstein E., Brassard G., Vazirani U. (1997). *The strengths and weaknesses of quantum computation.* SIAM Journal on Computing. 26 (5): 1510-1523.

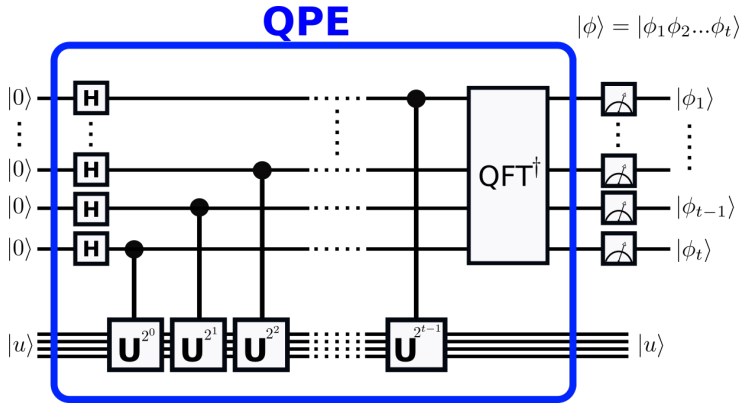

Figure 15: The rest of the quantum phase estimation (QPE) circuit for measuring the phase ϕ of an eigenvalue with t-bits of accuracy. Here $\text{QFT}^\dagger$ is the inverse quantum Fourier transform (IQFT). For t bits in the inverse Fourier transform, the circuit gives t measured bits for the phase ϕ of the eigenvalue of $\mathbf{U}$ that is associated with eigenvector $|u\rangle$.

Approximate $\phi \in [0, 1)$ as a binary number base 2

$$\phi \approx 0.\phi_1\phi_2\phi_3...\phi_t \tag{86}$$

$$= \sum_{i=1}^{t} \phi_i 2^{-i} \tag{87}$$

where ϕ_i are binary digits $\in \{0, 1\}$. Equation 85 becomes

$$|\psi\rangle = \frac{1}{2^{t/2}}(|0\rangle + e^{2\pi i 0.\phi_t)}\,|1\rangle) \otimes (|0\rangle + e^{2\pi i 0.\phi_{t-1}\phi_t)}\,|1\rangle) \otimes ....(|0\rangle + e^{2\pi i 0.\phi_1\phi_2...\phi_t)}\,|1\rangle) \otimes |u\rangle\,. \tag{88}$$

This is equivalent to the product representation of the Fourier transform which means that we can invert the state using an inverse Fourier transform,

$$\text{QFT}^\dagger \otimes I^n\,|\psi\rangle = |\phi_1\phi_2.....\phi_t\rangle \otimes |u\rangle\,. \tag{89}$$

where n is the number of bits in the lower set and that describes $|u\rangle$. Prior to measurement the state is that on the right hand side of equation 89. When we measure the top t bits we will have a t bit approximation for ϕ. The rest of the circuit, including inverse QFT and measurements, is shown in Figure 15.

6.2.1 Efficiency

How many calls to $\mathbf{U}$ are required for t bits of precision in the measurement of ϕ? The number of operations of $\mathbf{U}$ is $\sum_{j=0}^{t-1} 2^j = 2^t - 1$. How many additional gates are required?

The Hadamard are $O(t)$ gates. The QFT requires $O(t^2)$ gates. For $t \geq 5$, the number of gates in the series of controlled U operations dominates.

6.2.2 The QPE operator

Suppose we do not input a state that is a single eigenvector into the circuit? What does the QPE operator outlined in Figure 15 do? We describe $\mathbf{U}$ in terms of a basis of all its eigenvectors $\{u_j\}$

$$\mathbf{U} = \sum_j \lambda_j |u_j\rangle \langle u_j| = \sum_j e^{2\pi i \phi_j} |u_j\rangle \langle u_j| . \tag{90}$$

Here $\{\lambda_j\}$ refer to the set of eigenvalues and $\{\phi_j\}$ to their phases with $\lambda_j = e^{2\pi i \phi_j}$. We describe the input state in terms of a superposition of the eigenvector basis

$$|\psi\rangle = \sum_j a_j |u_j\rangle \tag{91}$$

The output of the quantum phase estimation (QPE) operator (as shown in Figure 15 for each eigenstate $|u_j\rangle$) is given by equation 89 and is

$$\mathbf{QPE} |0\rangle^{\otimes t} |u_j\rangle = |\phi_j\rangle \otimes |u_j\rangle . \tag{92}$$

Here $|\phi_j\rangle$ refers to the state given by t bits of phase ϕ_j. Applying this operation to the superposition state in equation 91

$$\mathbf{QPE} |0\rangle^{\otimes t} \sum_j a_j |u_j\rangle = \sum_j a_j |\phi_j\rangle \otimes |u_j\rangle . \tag{93}$$

This form for the QPE will be useful for other algorithms (such as the HHL for solving a linear equation; see section 7).

6.3 Quantum Counting Algorithms

In Grover's algorithm, it is difficult to chose the number of iterations without knowing the number of solutions. One approach repeats Grover's algorithm multiple times, choosing a random number of iterations in each run. While inelegant, this approach does succeed in finding a solution with high probability in $O(\sqrt{N})$ queries.

On a quantum computer it is possible to estimate the number of solutions much more quickly than is possible on a classical computer by combining the Grover iteration with the phase estimation technique. **Quantum counting**, allows us to determine whether a given search problem has any solutions, and to find one if there are, even if the number of solutions is not known in advance. Fourier transform based phase estimation procedure enables us to estimate M the number of solutions to high accuracy with $\mathrm{O}(\sqrt{N})$ queries.

Quantum counting is done as follows. Run the Grover iterator 1 time and also 2 times and also 4 times up to 2^{t-1} times. Each output controls a bit in a register. Then an inverse Fourier transform is performed on the t bit register which is then measured, as shown in the phase measurement circuit shown in Figure 15. The periodicity in equation 83 is detected which gives an estimate for the angle θ and so for the number of solutions (via equation 74).

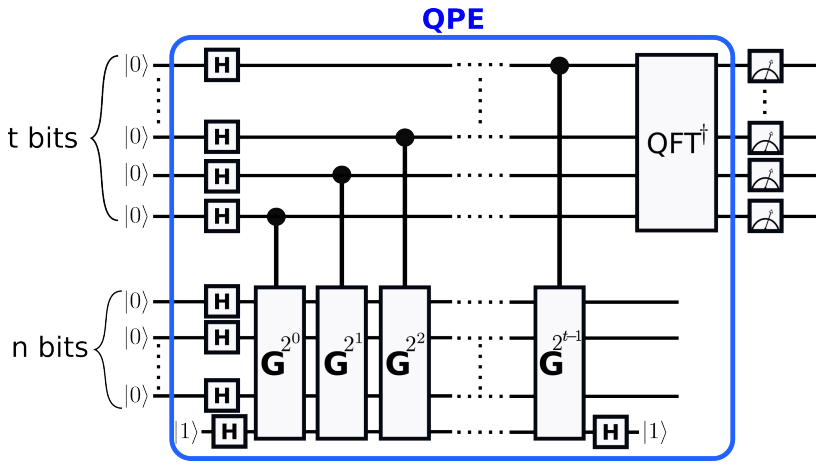

Figure 16: An algorithm for counting solutions. Here G is the Grover iterator. Here t is the number of bits used for phase estimation and n is the number of bits for U which lies within the Grover iterator. The bottom qubit is a work-bit used in the Grover iterator. The operation within the large box is the quantum phase estimation algorithm.

Recall that the quantum phase estimating circuit used as input an eigenvalue of the unitary operator. What are the eigenvalues of the Grover iteractor? The Grover iterator G depends on an angle θ which satisfies $\sin(\theta/2) = \sqrt{M/N}$ (equation 74) where M is the number of solutions and N is the number of states. In the basis of non-solutions $|v\rangle$ and solutions $|s\rangle$, the Grover iterator is $G = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$ (equation 82). The eigenvalues of the Grover iterator are

$$e^{i\theta} \qquad \text{and} \qquad e^{-i\theta}$$

with eigenvectors

$$|e+\rangle = \frac{1}{\sqrt{2}}(|v\rangle - i\,|s\rangle) \qquad \text{and} \qquad |e-\rangle = \frac{1}{\sqrt{2}}(|v\rangle + i\,|s\rangle).$$

We can write the initial state in terms of the eigenvalues

$$\begin{aligned}|y\rangle &= \cos(\theta/2)\,|v\rangle + \sin(\theta/2)\,|s\rangle \\ &= \frac{1}{\sqrt{2}}(e^{i\theta/2}\,|e+\rangle + e^{-i\theta/2}\,|e-\rangle).\end{aligned}$$

Our input to the quantum circuit in Figure 16 is a superposition of two eigenvectors. That means that the Fourier transform will contain a superposition of two different phases. When the phase is measured, one of these two eigenvalue phases will be measured. It should be possible to tell the two possibilities apart as they are $\theta, -\theta$. Note that the phase estimation circuit contains a factor of 2π in the definition of the eigenvalue, so after you take your measurement of the phase ϕ you should multiply it by 2π to find θ.

6.4 3-SAT

We discuss 3-SAT problems because Grover's algorithm can be used to efficiently find their solutions.

3-SAT is a Boolean satisfiability problem (a decision problem) where each clause consists of three Boolean variables. For example on a set of n Boolean variables $\mathbf{x} = (x_1, x_2, x_3, x_4....x_n)$ with $x_i \in \{0, 1\}$ a 3-SAT problem consists of a logical expression which looks like

$$\begin{aligned}\Phi(\mathbf{x}) =&(x_1 \vee x_2 \vee \bar{x}_7) \wedge (\bar{x}_2 \vee \bar{x}_8 \vee x_9) \wedge \\ &(x_2 \vee \bar{x}_5 \vee x_7) \wedge (x_1 \vee \bar{x}_5 \vee \bar{x}_9) \wedge \\ &() \wedge ().....\end{aligned} \tag{94}$$

Each clause looks like this: $x_1 \vee x_2 \vee \bar{x}_7$ (corresponding to x_1 OR x_2 OR NOT (x_7)) and contains 3 Boolean variables. Each variable within the clause is either a Boolean variable x_i or its opposite ($\bar{x}_i$ = NOT (x_i)). All clauses must be satisfied ($\wedge$ is equivalent to AND). The function Φ returns a Boolean (either 0 or 1). The goal is to determine if there are solutions (one or more possible binary strings $\mathbf{x}$) giving $\Phi(\mathbf{x}) = 1$. The form of equation 94 is called the **conjunctive normal form**.

Any combinatorial logic circuit can be converted efficiently (with fewer than an exponential number of terms) to a conjunctive normal form via the **Tseytin transformation** `https://en.wikipedia.org/wiki/Tseytin_transformation`. To reduce the unrestricted SAT problem to 3-SAT, one can transform each clause with n terms to a conjunction of $n - 2$ clauses. Here is a recipe:

$$\begin{aligned}(x_1 \vee x_2 \vee x_3 ... \vee x_k) \rightarrow &(x_1 \vee x_2 \vee w_2) \wedge \\ &(\bar{w}_2 \vee x_3 \vee w_3) \wedge \\ &(\bar{w}_3 \vee x_4 \vee w_4) \wedge ... \wedge \\ &(\bar{w}_{k-3} \vee x_{k-2} \vee w_{k-2}) \wedge \\ &(\bar{w}_{k-2} \vee x_{k-1} \vee x_k)\end{aligned}$$

The two relations are not logically identical but they are logically satisfiable. An additional set of $k-3$ auxiliary logical variables $w_2 ... w_{k-2}$ are used. A shorter example

$$(x_1 \vee x_2 \vee x_3 \vee x_4) \rightarrow (x_1 \vee x_2 \vee w) \wedge (\bar{w} \vee x_3 \vee x_4).$$

If $w = 1$ then we find x_1 or x_2 must be 1 for the expression to be true. If $w = 0$ then x_3 or x_4 must be 1 for the expression to be true. Taking into account both possible w values, the expression is satisfied/true if x_1 or x_2 or x_3 or x_4 is 1, as given in the original expression.

K-SAT, where each clause has k variables, is an NP problem. Any instance $\Phi(\mathbf{x})$ of the K-SAT problem with $\mathbf{x} \in \{0,1\}^n$ requires doing the computation of the Boolean function for 2^n different combinations of binary values for the input Boolean string $\mathbf{x}$. (This establishes that it takes exponential time to find solutions.) The correctness of any particular solution, i.e. given any proposed set of values for a binary string can be checked in polynomial time. That's because the expression itself is comprised of a polynomial number of variables and logical operations. (This establishes that it takes polynomial time to check solutions). For $K \geq 3$, K-SAT is proved to be NP-complete (Cook(1971) and Levin(1973); - that means all other NP problems can be converted into 3-SAT problems).

6.5 Solving a 3-SAT problem with Grover's algorithm

How do you run a 3-SAT problem on a quantum computer? Since 3-SAT gives a Boolean function $f(\mathbf{x})$, we can use the function $f()$ to construct a unitary oracle operator U_f (in equation 64) that can be incorporated within the Grover iterator. We repeat equation 64 here:

$$U_f \left| x, q \right\rangle \rightarrow \left| x, f(x) + q \right\rangle . \tag{95}$$

For example, suppose we would like to find solutions to

$$f(\mathbf{x}) = (x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee x_3 \vee \bar{x}_4). \tag{96}$$

It is convenient to use this relation

$$(a \vee b \vee c) = \overline{\bar{a} \wedge \bar{b} \wedge \bar{c}}$$

Equation 114 becomes

$$f(\mathbf{x}) = \overline{\bar{x}_1 \wedge \bar{x}_2 \wedge \bar{x}_3} \wedge \overline{\bar{x}_2 \wedge \bar{x}_3 \wedge x_4}. \tag{97}$$

A circuit that performs U_f (the oracle in equation 95), with $f(x)$ in equation 114 or 97, is shown in Figure 17. The circuit is not optimally efficient but illustrates that any 3-SAT problem can be implemented on a quantum computer and solved via Grover's algorithm. Grover's iterator needs to be run $O(\sqrt{N})$ times where the number of possible solutions $N = 2^n$ and n is the number of input variables (the length of the binary string $\mathbf{x}$) in the 3-SAT problem. Thus Grover's algorithm beats classical methods of searching for solutions

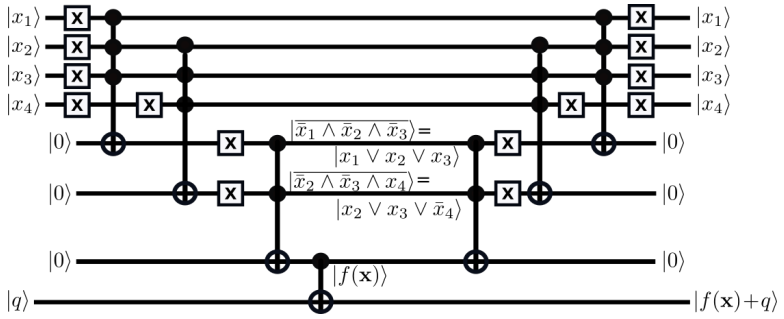

Figure 17: A circuit to perform the unitary operation U_f (the oracle) needed for the Grover iterator and for the 3-SAT function $f()$ in equation 114 or 97. I have tried to unentangle the 3 work bits so that they can be reused during the iteration.

which are $O(N) = O(2^n)$. (As $O(\sqrt{N}) = O(2^{n/2})$ is still exponential in n, Grover's algorithm is faster but not exponentially faster. It's not polynomial in n.).

I have illustrated one way to implement 3-SAT on a quantum computer. There is a nice 3-SAT example given in a *qisket* tutorial `https://qiskit.org/`, a quantum algorithm that leverages classical algorithms such as the Davis-Putnam-Logemann-Loveland algorithm (Zhang+20) and a google patent involving using density matrices? The research setting seems active.

7 Solving linear equations on a quantum computer (the HHL algorithm)

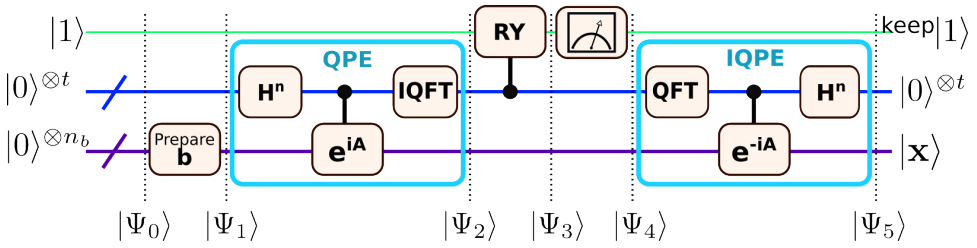

Figure 18: Illustrating the HHL algorithm.

Another application of the quantum phase estimation (QPE) algorithm (as introduced in section 6.2) is with the HHL algorithm for solving a linear equation. The goal is to efficiently solve a linear equation on a quantum computer. Given an N_b dimensional vector $\mathbf{b}$, and an $N_b \times N_b$ invertible square matrix $\mathbf{A}$ the goal is to find the solution to

$$\mathbf{A}\mathbf{x} = \mathbf{b}. \tag{98}$$

The quantum algorithm is known as HHL (Harrow-Hassidim-Lloyd)[10]. For an introduction to the algorithm see[11] Morrel Jr., H. J., A. Zaman, Wong, H. Y. (2022) *Step-by-Step HHL Algorithm Walkthrough to Enhance the Understanding of Critical Quantum Computing Concepts*, `https://arxiv.org/pdf/2108.09004.pdf`. We will follow this clear introduction here.

We assume that $\mathbf{A}$ is Hermitian. If $\mathbf{A}$ is not Hermitian then the system can be replaced with the following one where $\mathbf{A}'$ is Hermitian;

$$\mathbf{A}' = \begin{pmatrix} 0 & \mathbf{A} \\ \mathbf{A}^\dagger & 0 \end{pmatrix} \qquad \mathbf{b}' = \begin{pmatrix} 0 \\ \mathbf{b} \end{pmatrix} \qquad \mathbf{x}' = \begin{pmatrix} \mathbf{x} \\ 0 \end{pmatrix}.$$

We assume that $N_b = 2^{n_b}$ where n_b is the number of qubits needed to code $\mathbf{b}$. In terms of the eigenvectors $\{|a_j\rangle\}$ and eigenvalues $\{\lambda_j\}$ of $\mathbf{A}$

$$\mathbf{A} = \sum_{j=0}^{N_b-1} \lambda_j |a_j\rangle \langle a_j| . \tag{99}$$

In the same basis the vector $\mathbf{b}$ is

$$|b\rangle = \sum_{j=0}^{N_b-1} \beta_j |a_j\rangle . \tag{100}$$

To put the problem on a quantum computer this state should be normalized so that it has length 1, so we require that $\sum_j |\beta_j|^2 = 1$. We would like to find the solution to equation 98 which is

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b} = \sum_{j=0}^{N_b-1} \lambda_j^{-1} \beta_j |a_j\rangle . \tag{101}$$

This too should be normalized! The state that is measured would be proportional to the actual solution so that it is normalized. It will be convenient to use the unitary operator

$$\mathbf{U} = e^{i\mathbf{A}} = \sum_{j=0}^{N_b-1} e^{i\lambda_j} |a_j\rangle \langle a_j| . \tag{102}$$

We work with three registers of qubits, as shown in Figure 18. We order the registers $\mathcal{H}_a \otimes \mathcal{H}_t \otimes \mathcal{H}_b$ where $\mathcal{H}_a$ has a single qubit, $\mathcal{H}_t$ has t qubits and $\mathcal{H}_b$ has n_b qubits. We start with

$$|\Psi_0\rangle = |1\rangle_a |0\rangle_t^{\otimes t} |0\rangle_b^{\otimes n_b} .$$

[10] A. W. Harrow, A. Hassidim, and S. Lloyd, *Quantum Algorithm for Linear Systems of Equations*, Physical Review Letters, 103 (2009), no. 15, 150502, `https://arxiv.org/abs/0811.3171`.

[11] Morrel Jr., H. J., A. Zaman, Wong, H. Y. (2022) *Step-by-Step HHL Algorithm Walkthrough to Enhance the Understanding of Critical Quantum Computing Concepts*, `https://arxiv.org/pdf/2108.09004.pdf`.

The vector **b** is prepared and placed in the b register

$$|\Psi_1\rangle = |1\rangle_a |0\rangle_t^{\otimes t} |\mathbf{b}\rangle_b = |1\rangle_a |0\rangle_t^{\otimes t} \sum_{j=0}^{N_b-1} \beta_j |a_j\rangle_b . \tag{103}$$

The quantum phase estimation (QPE) circuit executed on the b and t registers consists of a t-bit Hadamard, followed by a series of t controlled **U** gates to different powers of 2, followed by a t-bit inverse quantum Fourier transform (IQFT). For an input state $\sum_j a_j |u_j\rangle$, The QPE circuit does this

$$\mathbf{QPE} |0\rangle^{\otimes t} \sum_{j=0}^{N_b-1} a_j |u_j\rangle = \sum_j a_j |\phi_j\rangle \otimes |u_j\rangle . \tag{104}$$

(see discussion above in section 6.2 and equation 93). Here each $|u_j\rangle$ is an eigenvector of the **U** operator with eigenvalue $e^{2\pi i \phi_j}$ and the state $|\phi_j\rangle$ is a quantum state specified by t bits of accuracy in the value of ϕ_j itself. Note that because $\mathbf{U} = e^{i\mathbf{A}}$, the eigenvalues of **A** are related to the phases of the eigenvalues of **U** via

$$\lambda_j = 2\pi\phi_j. \tag{105}$$

For one of the phases $\phi = 0.j_1 j_2 ... j_t$ (via powers of 1/2) and $j_1, j_2, ...j_t \in \{0, 1\}$. The state associated with phase ϕ would be $|\phi\rangle = |j_1 j_2 .... j_t\rangle$. The bits $j_1, j_2 ... j_t$ are related to the value of ϕ via $\phi = 2^{-j_1} + 2^{-2j_2} + ... + 2^{-tj_t}$ so that $\phi \in [0, 1)$. Each phase has its own fractional expansion!

We apply the QPE operator to $|\Psi_1\rangle$ in equation 103 giving

$$\begin{aligned} |\Psi_2\rangle &= \mathbf{QPE} |\Psi_1\rangle \\ &= |1\rangle_a \otimes \sum_{j=0}^{N_b-1} \beta_j |\phi_j\rangle_t |a_j\rangle_b . \end{aligned} \tag{106}$$

Note we are labelling the states according to where they are in the circuit illustrated in Figure 18.

7.0.1 The controlled rotation

In the next step we use a controlled rotation that targets the lone ancillary qubit. This is the hard part! The rotation angle $\alpha(\phi)$ is controlled so that it depends on the eigenvalue phase ϕ which is located in the t-register,

$$\mathbf{RY}(\alpha(\phi)) = \begin{pmatrix} \cos\alpha(\phi) & \sin\alpha(\phi) \\ -\sin\alpha(\phi) & \cos\alpha(\phi) \end{pmatrix} . \tag{107}$$

We illustrate the rotation on a single phase $|\phi\rangle$

$$\mathbf{RY}\,|1\rangle_a \otimes |\phi\rangle_t = [\sin(\alpha(\phi))\,|0\rangle_a + \cos(\alpha(\phi))\,|1\rangle_a]\,|\phi\rangle_t \tag{108}$$

Applying the rotation to $|\Psi_2\rangle$ in equation 106 gives

$$\begin{aligned} |\Psi_3\rangle &= \mathbf{RY}\,|\Psi_2\rangle \\ &= \sum_{j=0}^{N_b-1} [\sin(\alpha(\phi_j))\,|0\rangle_a + \cos(\alpha(\phi_j))\,|1\rangle_a]\beta_j\,|\phi_j\rangle_t\,|a_j\rangle_b\,. \end{aligned} \tag{109}$$

The rotation is described as a y-rotation because it rotates the qubit about the y axis on the Bloch sphere.

The lone ancillary bit is measured and the entire calculation discarded unless the measurement gives $|1\rangle_a$. The resulting state is

$$|\Psi_4\rangle = \frac{1}{\sqrt{\sum_{k=0}^{N_b-1} |\beta_k \cos(\alpha(\phi_k))|^2}}\,|1\rangle_a \otimes \sum_{j=0}^{N_b-1} \cos(\alpha(\phi_j))\beta_j\,|\phi_j\rangle_t\,|a_j\rangle_b\,. \tag{110}$$

We apply the inverse QPE circuit giving

$$\begin{aligned} |\Psi_5\rangle &= \mathbf{IQPE}\,|\Psi_4\rangle \\ &= \frac{1}{\sqrt{\sum_{k=0}^{N_b-1} |\beta_k \cos(\alpha(\phi_k))|^2}}\,|1\rangle_a \otimes |0\rangle_t^{\otimes t} \otimes \sum_{j=0}^{N_b-1} \cos(\alpha(\phi_j))\beta_j\,|a_j\rangle_b\,. \end{aligned} \tag{111}$$

The b-register is no longer entangled with the other two registers. The resulting state in the b-register is

$$|\psi\rangle_b = \frac{1}{\sqrt{\sum_{k=0}^{N_b-1} |\beta_k \cos(\alpha(\phi_k))|^2}} \sum_{j=0}^{N_b-1} \cos(\alpha(\phi_j))\beta_j\,|a_j\rangle_b \tag{112}$$

This resembles the solution for $\mathbf{x}$ in equation 101 if we can chose the angle function $\alpha(\phi)$ so that

$$\cos(\alpha(\phi_j)) \propto \frac{1}{2\pi\phi_j} = \frac{1}{\lambda_j} \tag{113}$$

or with rotation angle

$$\alpha(\phi) = \operatorname{acos} \frac{C}{2\pi\phi} \tag{114}$$

where C is a coefficient that can be adjusted. (It might make sense to choose $C = 2\pi$ as $\phi \in [0, 1)$.) Using this kind of function for α, equation 111 becomes

$$|\Psi_5\rangle \propto |1\rangle_a \otimes |0\rangle_t^{\otimes t} \otimes \sum_{j=0}^{N_b-1} \frac{\beta_j}{\lambda_j} |a_j\rangle_b \tag{115}$$

which is the desired solution!

Implementing the function in equation 114 seems non-trivial! Nevertheless there have been a number of successful experimental implementations of the HHL algorithm, so it must be possible! I am curious as to how such a functions can be implemented.

Measurement of the b-register at the end of the circuit gives an approximation for the solution $\mathbf{x}$. Why is it an approximation? The accuracy of the solution depends upon the number of bits in the t-register which controls the accuracy of the phase estimation.

The vector $\mathbf{b}$ need not be input in any particular basis as long as it is consistent with that used for $\mathbf{A}$ and $\mathbf{U}$ and that expected for the solution $\mathbf{x}$.

7.1 Comments on variants

A variant of the HHL algorithm could be used to create operators that involve computing more interesting functions of the eigenvalues of an operator $\mathbf{A}$. Here the function computed with the $\mathbf{RY}$ operator is $f(\lambda) = \lambda^{-1}$ and it inverts the eigenvalues. With $\mathbf{A} = \mathbf{V}\Lambda\mathbf{V}^\dagger$, the singular value decomposition of $\mathbf{A}$, $\mathbf{V}$ unitary and Λ a diagonal matrix containing the eigenvalues of $\mathbf{A}$, we could use a similar algorithm to compute the operator $\mathbf{B} = \mathbf{V}f(\Lambda)\mathbf{V}^\dagger$ where $f(\Lambda)$ is a more complicated function of the eigenvalues. To create the operator $\mathbf{B}$ we would need to construct an operator $\mathbf{RY}$ with rotation angle $\alpha(\phi)$ giving $\cos(\alpha(\phi)) = f(2\pi\phi)$.

Can the HHL algorithm be used to amplify low energy states? For example if $\mathbf{A} = e^{i\mathbf{H}}$ and $|b\rangle$ gives an initial guess for the ground state of $\mathbf{H}$, a couple iterations of the HHL algorithm would give a state vector that is closer to the ground state. Iterating the entire algorithm would be inefficient because it would require discarding the $|0\rangle_a$ measurements numerous times. Perhaps one could create a steeper version of the controlled rotation (implementing $e^{-\lambda}$ instead of $1/\lambda$) which would be equivalent to iterating.

8 Adiabatic Algorithms

8.1 Finding the ground state of a complex Hamiltonian via adiabatic evolution

Adiabatic algorithms are based on what is called the **adiabatic theorem** in quantum mechanics.[12]

[12] See `https://en.wikipedia.org/wiki/Adiabatic_theorem`.

Suppose you have an initial Hamiltonian H_i and you know its ground state $|0\rangle$. You would like to know the ground state of another, similar problem (or final) Hamiltonian H_f. Construct a time dependent Hamiltonian

$$H(t) = H_i(1 - s(t)) + H_f s(t). \tag{116}$$

where $s(t) \in [0, 1]$ is a smooth, slowly increasing function that starts at $s(0) = 0$ and ends at 1, with $s(T) = 1$ at some later time T.

With $\hbar = 1$, a quantum state vector $|\psi(t)\rangle$ evolves as

$$\frac{d\,|\psi(t)\rangle}{dt} = -iH(t)\,|\psi(t)\rangle\,. \tag{117}$$

The wave function slowly and continuously evolves from the initial ground state $|\psi(0)\rangle$ with the time dependent Hamiltonian

$$|\psi(t)\rangle = e^{-i\int_0^t dt'\ H(t')}\,|\psi(0)\rangle\,. \tag{118}$$

If the evolution is done sufficiently slowly, then the adiabatic theorem states that the final wave vector $|\psi(T)\rangle$ will be very close to the ground state of the problem Hamiltonian H_f. Gradual change from one ground state brings the system to another ground state.

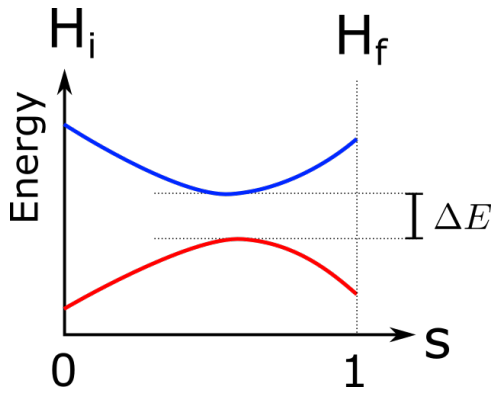

Figure 19: We show the evolution of the two lowest eigenvalues as the parameter s is slowly varied. The Hamiltonian begins as H_i when $s = 0$ and becomes H_f when $s = 1$. The system begins in the ground state of H_i. The system must evolve slowly for it to remain in the ground state. The time needed to pass the region where the energy gap is small depends on the width of the energy gap ΔE.

What makes an adiabatic algorithm difficult?

If at some point during the evolution the two lowest energy eigenvalues of the system are close to each other then the system can undergo a transition out of the ground state. To prevent this, you would need to evolve the system very slowly, or *adiabatically*. To pass

the region sufficiently adiabatically to suppress transition out of the ground state, the drift rate δ (in units of energy/time) must be slower than[13]

$$\delta \lesssim (\Delta E)^2 \tag{119}$$

where ΔE is the minimum energy gap between the two energy eigenstates. Note that with $\hbar = 1$ energy is in units of frequency and the drift rate is in units of frequency squared. The time for each operation step $\tau \sim E/\delta$ gives

$$\tau \gtrsim \frac{E}{(\Delta E)^2} \tag{120}$$

where E is a characteristic energy at the transition where the two energy eigenvalues are the same.

Qualitatively Rabi oscillations in a two state system take place at a frequency given by ΔE. To evolve slowly enough to remain in the adiabatic regime, the time for the energy to cross a distance ΔE at drift rate δ should be longer than a Rabi oscillation period. To order of magnitude this gives equation 119.

Do adiabatic algorithms actually work? It is not necessarily easy in advance to determine the difference in energy between the ground state and first excited state as a function of time during an adiabatic evolution calculation. This makes it challenging to efficiently choose the time step.

One could run the adiabatic computation more than once but with different initial Hamiltonians and initial ground states. The hope would be that one of the choices for the initial Hamiltonian would be less prone to problems associated with proximity in the lowest two energy levels during the adiabatic calculation. I have noticed that people do not tend to choose an initial Hamiltonian with the simplest possible ground state, $|0\rangle^{\otimes n}$. An initial ground state prepared with the n-bit Hadamard $H^n |0\rangle^{\otimes n}$ seems more popular, though I am not sure why.

Another approach is to introduce some intermediate Hamiltonians within the adiabatic variation;

$$H(s) = A(s)H_i + \sum_j B_j(s)H_j + C(s)H_f. \tag{121}$$

The time dependent functions $A(s), B(s), C(s)$ are designed so that the system moves from the ground state of the initial Hamiltonian H_i to ground states of the intermediate Hamiltonians (here denoted H_j) before approaching ground state of the desired final Hamiltonian

[13]The Landau-Zener formula is an analytic solution to the equations of motion governing the transition dynamics of a two-state quantum system, with a time-dependent Hamiltonian varying such that the energy separation of the two states is a linear function of time. The formula gives the probability of a *diabatic* (not adiabatic) transition between the two energy states and it is exponentially dependent upon the drift rate. See C. De Grandi and A. Polkovnikov 2009, *Adiabatic perturbation theory: from Landau-Zener problem to quenching through a quantum critical point*, in the book: Quantum Quenching, Annealing and Computation, pp 75-114, `https://arxiv.org/pdf/0910.2236.pdf`.

H_f. This provides some flexibility in the algorithm which could help improve and optimize it.

Variational techniques which involve both classical and quantum operations, may have advantages over adiabatic algorithms. We will discuss problems that could be solved with an adiabatic algorithm, but keep in mind that many of the components of these problems might also be useful for application in variational quantum algorithms.

8.2 Quantum adiabatic optimization

Quantum adiabatic optimization is a class of procedures for solving optimization problems using a quantum computer.[14]

- Design a problem Hamiltonian H_f whose ground state encodes the solution of an optimization problem.
- Prepare the known ground state of a simple Hamiltonian, H_i.
- Interpolate adiabatically by approximating the evolution with series of quantum operations.
- Measure quantities of the ground state of H_f.

Why is this an optimization problem? This is because the ground eigenstate of H_f is the wave function that minimizes $\langle\psi| H_f |\psi\rangle$.

8.3 Trotterization

In many physical systems energy states and evolution can be approximated with a sum of Hamiltonians. For example with one Hamiltonian representing the energy in a subsystem and another Hamiltonian representing interactions between the subsystem and external degrees of freedom. Let $iH = A + B$. However the operators or matrices A and B need not commute. In other words $AB \neq BA$ and that means that $e^{A+B} \neq e^A e^B$. Let us use a small parameter ϵ,

$$e^{A+B} \approx e^{\epsilon A} e^{\epsilon B} e^{\epsilon A} e^{\epsilon B} ...$$

where we repeat pairs of $e^{\epsilon A} e^{\epsilon B}$ a total number of $r = 1/\epsilon$ times. Why is this a good approximation? If ϵ is small then $e^{\epsilon A} \sim I + \epsilon A$ and

$$\begin{aligned} &e^{\epsilon A} e^{\epsilon B} e^{\epsilon A} e^{\epsilon B} ... \sim (I + \epsilon A)(I + \epsilon B)(I + \epsilon A)(I + \epsilon B).... \\ &\sim I + (A + B). \end{aligned}$$

To first order this is equivalent to $e^{A+B} \sim I + A + B + ...$

[14] Proposed in the context of quantum computation by *Quantum Computation by Adiabatic Evolution*, Farhi, Goldstone, Gutmann, and Sipser (2000) `https://arxiv.org/abs/quant-ph/0001106`.

Equivalently Trotterization is the approximation

$$e^{A+B} \sim \left(e^{\frac{A}{r}} e^{\frac{B}{r}}\right)^r. \tag{122}$$

This means we can apply the operator $e^{\frac{A}{r}} e^{\frac{B}{r}} \sim (I + \frac{A}{r})(I + \frac{B}{r})$ a total number of r times to approximate e^{A+B}. How big is the error? This is a first order approximation which is accurate $O(1/r)$. In other words

$$\|e^{A+B} - (e^{\frac{A}{r}} e^{\frac{B}{r}})^r\| \leq \frac{1}{r}.$$

Totterization is in general useful for approximating an operator that is an exponential of a sum of simpler operators, as in equation 122. With r large, each piece is to a good approximation a first order expression. Or the operators $e^{\epsilon A}$ and $e^{\epsilon B}$ could be simple unitary operators which can be straightforward to implement even if $e^{(A+B)}$ is difficult to calculate. A similar approach is used to derive symplectic integrators in the setting of celestial mechanics. In this setting the Hamiltonian can be divided into more than two pieces with potential interaction terms in one piece and Keplerian terms for single bodies in another one.

Going back to equation 116 for the adiabatic evolution of a Hamiltonian system from $H_i \to H_f$ which we repeat here,

$$H(t) = H_i(1 - s(t)) + H_f s(t). \tag{123}$$

we assume that $s = t/T$ and discretize the transition from $s = 0$ to 1 into r steps. We index each step with index $j \in [1, r]$, giving $s_j = j/r$ and $t_j = s_j T$. This gives Hamiltonian at time step j

$$H_j = H(t_j) = H_i \left(1 - \frac{j}{r}\right) + H_f \frac{j}{r}. \tag{124}$$

The time step has duration $dt = T/r$, so the exponential operator we apply at the j-th step is

$$V_j = e^{-iH_j dt} = e^{-i\left(H_i\left(1-\frac{j}{r}\right)+H_f\frac{j}{r}\right)\frac{T}{r}}. \tag{125}$$

A Trotterized version of V_j is

$$V_j \approx e^{-iH_i\left(1-\frac{j}{r}\right)\frac{T}{r}} e^{-iH_f\frac{j}{r}\frac{T}{r}}. \tag{126}$$

The entire sequence of r operations for the adiabatic evolution from $t = 0 \to T$ is approximated as

$$\begin{aligned} e^{-i\int_0^T dt\ H(t)} &\approx V_r ....V_3 V_2 V_1 \\ &\approx \Pi_{j=1}^r e^{-iH_i\left(1-\frac{j}{r}\right)\frac{T}{r}} e^{-iH_f\frac{j}{r}\frac{T}{r}}. \end{aligned} \tag{127}$$

The order of the operators in the product is important. The first operator applied should be that with $j = 1$. This expression can be used to evolve the ground state of H_i, to that of H_f as in equation 118.

9 Optimization problems

9.1 Optimization of quadratic Boolean functions and the Ising model

A **quadratic unconstrained pseudo-Boolean optimization** (QUBO) problem involves a real function of the form

$$f_{ue}(\mathbf{x}) = \sum_i u_i x_i + \sum_{i<j} e_{ij} x_i x_j, \tag{128}$$

where $\mathbf{x} = (x_1, x_2, ...x_n)$, is a Boolean string $x_i \in \{0, 1\}$ and u_i, e_{ij} are real coefficients. The function f_{ue} returns a real number. The goal is to find a Boolean string $\mathbf{x}$ that minimizes the function $f_{ue}(\mathbf{x})$. QUBO is **NP-hard** because 2^n possible solutions must be tried to measure f. However, an optimization problem is not a decision problem. Optimization problems are *not in NP* because they are not decision problems. You can compute a function f for a particular $\mathbf{x}$ in polynomial time, but you would not know necessarily that you have a minimum.

The function in equation 128 has the same minimum as the function

$$g_{ue}(\mathbf{s}) = \sum_i u'_i s_i + \sum_{i<j} e'_{ij} s_i s_j + \text{constant}, \tag{129}$$

where $\mathbf{s}$ is a string of values with each $s_i \in \{1, -1\}$. Here

$$u'_i = u_i/2 + \sum_j (e_{ij} + e_{ji})/4 \tag{130}$$

$$e'_{ij} = e_{ij}/4 \tag{131}$$

can be computed from the u_i and e_{ij} coefficients in equation 128 and

$$x_i = (1 + s_i)/2 \qquad \text{equivalently} \qquad s_i = 2x_i - 1. \tag{132}$$

A quadratic Boolean optimization (QUBO) model is related to an **Ising model** via variable substitution

$$x_i \to (I + \sigma_{z,i})/2 \qquad \text{or} \qquad s_i \to \sigma_{z,i}. \tag{133}$$

The result is a Hamiltonian for n qubits

$$H_{ue} = \sum_i u'_i \sigma_{z,i} + \sum_{i<j} e'_{ij} \sigma_{z,i} \sigma_{z,j} + \text{constant} \tag{134}$$

This Hamiltonian is related to an ensemble of spin systems perturbed by a magnetic field (giving the u'_i coefficients) and that interact (giving the e'_{ij} coefficients). A Hamiltonian

that only depends on $\sigma_{z,i}$ operators and contains local binary interaction terms is known as an **Ising model**.

Thus optimization of the quadratic optimization problem in equation 128 (finding a minimum) is related to finding the ground state of the Ising model Hamiltonian in H_{ue}. In fact the ground state of the Ising Hamiltonian H_{ue} must be a basis state in the conventional basis with states that look like $|0010100111...\rangle$. This is because the Hamiltonian commutes with $\sigma_{z,i}$ for every i. We say that H_{ue} and $\{\sigma_{z,i}\}$ are *simultaneously diagonalizable*. That means that eigenvectors of H are also eigenstates of $\sigma_{z,i}$. Because the ground state of the Ising model must be a pure state (rather than a superposition) in the conventional basis, a measurement of the ground state in the conventional basis gives a solution to the associated quadratic Boolean optimization (QUBO) problem.

In summary, finding the minimum in a quadratic Boolean optimization problem is equivalent to a finding the ground state of an Ising model.

9.2 Quadratization – reducing a polynomial Boolean optimization problem to a quadratic one

Supposing you would like to find the minimum of a Boolean function that contains cubic or higher order terms? At the expense of adding more variables, it is possible to reduce a higher order polynomial optimization problem to a quadratic one (which then can be implemented on a quantum computer via the Ising model). The procedure is known as **degree reduction** or **quadratization**. For a summary of methods for degree reduction see the useful, comprehensive and detailed review[15] by N. Dattani (2019) *Quadratization in Discrete Optimization and Quantum Mechanics*, `https://arxiv.org/pdf/1901.04405.pdf`.

Suppose we have an optimization problem where we would like to find a minimum of the function

$$f(\mathbf{x}) = \sum_i a_i x_i + \sum_{ij} a_{ij} x_i x_j + \sum_{ijk} a_{ij} x_i x_j x_k + ... \tag{135}$$

with Boolean variables $\mathbf{x} \in \{0,1\}^n$, coefficients $a_{ij...}$ are real, and with $f()$ returning a real number. Suppose the function $f()$ has maximum number of variables in any term equal to positive integer d. The integer d is the **degree** of the optimization function. We can also write equation 135 as

$$f_d(\mathbf{x}) = \sum_S \alpha_S \prod_{j \in S} x_j \tag{136}$$

where each S is a string of at most d distinct positive integer indices in order of size, and with each index an integer ranging from 1 to n where n is the number of input variables (the

[15] N. Dattani (2019) *Quadratization in Discrete Optimization and Quantum Mechanics*, `https://arxiv.org/pdf/1901.04405.pdf`

length of $\mathbf{x}$). Each string $S = k_1, k_2, k_3...k_d$ with order $k_1 < k_2... < k_d$ and $k_i \in \{1, 2, ..., n\}$. As there are at most d variables in each product, the function is a d-degree polynomial.

It is possible to find another function, $g(\mathbf{x}, \mathbf{w})$ that depends upon additional Boolean variables $\mathbf{w}$ that has the same minimum but is lower degree than f_d.

The optimization problem of a d-degree polynomial can be reduced to a *quadratic* (degree 2) optimization problem. The reduction can be done in polynomial time and the size (the number of terms) of the final polynomial can be bounded (dependent on d).

9.2.1 Freedman method for negative terms

Negative terms are relatively straightforward! We describe a popular method known as the **Freedman** method.[16] Consider a cubic term with a negative coefficient. With the addition of a new Boolean variable w

$$-x_1x_2x_3 = \min_{w\in\{0,1\}}[w(2 - x_1 - x_2 - x_3)]. \tag{137}$$

Recall that each variable x_i is either 0 or 1. The left hand side is only a minimum if all variables are 1 and in that case the product $x_1x_2x_3 = -1$. If any of the variables is not 1 then the product is 0. The right hand side has a minimum if $x_1 = x_2 = x_3 = w = 1$ and in that case the function gives -1. If one or more of the variables x_1, x_2, x_3 is zero, then $(2 - \sum_i x_i) \geq 0$ and function gives 0 with $w = 0$. Thus the left hand side is equivalent to the right hand side. This can be generalized; for a d degree term ;

$$-\prod_{i=1}^{d} x_i = \min_{w\in\{0,1\}}[w(d - 1 - \sum_{i=1}^{d} x_i)]. \tag{138}$$

The method reduces a negative coefficient term of any degree to a sum of quadratic terms with the addition of a single auxiliary Boolean variable w.

Example:
$H_6(\mathbf{x}) = -2x_1x_2x_3x_4x_5x_6 + x_5x_6$
becomes
$H_6(\mathbf{x}, w) = 2w(5 - x_1 - x_2 - x_3 - x_4 - x_5 - x_6) + x_5x_6$

9.2.2 Bit flipping for positive terms

Methods for reducing positive terms are more complex and usually involve more auxiliary variables and larger numbers of terms in the resulting quadratic expression. We describe an iterative method by Ishikawa.[17]

[16] D. Freedman and P. Drineas, *Energy Minimization via Graph Cuts: Settling What is Possible*, in 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), Vol. 2 (IEEE, 2005) pp. 939-946.

[17] H Ishikawa, *Transformation of General Binary MRF Minimization to the First-Order Case,* IEEE Transactions on Pattern Analysis and Machine Intelligence 33, 1234-1249 (2011).

We start with

$$\prod_{i=1}^{d} x_i = x_1 x_2 ... x_d$$

for a positive d degree term. We use NOT of one of the variables; $\bar{x}_1 = 1 - x_1$.

$$\prod_{i=1}^{d} x_i = (1 - \bar{x}_1) x_2 .. x_d = x_2 x_3 ... x_d - \bar{x}_1 x_2 .... x_d$$

We are left with a positive term that has degree $d-1$ and a negative term with degree d. The negative term can be treated with the Freedman method as described in the previous section. The $d-1$ degree positive term can be reduced by using the NOT of the first variable in its product and proceeding iteratively. Note that if you choose to replace $\bar{x}_i$ with a new variable, you must replace it throughout the logical expression.

Example:

$$\begin{aligned}
H(\mathbf{x}) &= x_1 x_2 x_3 x_4 \\
&= (1 - \bar{x}_1) x_2 x_3 x_4 \qquad \text{flip } x_1 \\
H(\mathbf{x}, w_1) &= x_2 x_3 x_4 + w_1 (3 - \bar{x}_1 - x_2 - x_3 - x_4) \qquad \text{use Freedman method} \\
&= (1 - \bar{x}_2) x_3 x_4 + w_1 (3 - \bar{x}_1 - x_2 - x_3 - x_4) \quad \text{flip } x_2 \text{ in positive term} \\
H(\mathbf{x}, w_1, w_2) &= x_3 x_4 + w_2 (2 - \bar{x}_2 - x_3 - x_4) + w_1 (3 - \bar{x}_1 - x_2 - x_3 - x_4) \quad \text{Freedman method} \\
&= x_3 x_4 + w_2 (1 + x_2 - x_3 - x_4) + w_1 (2 + x_1 - x_2 - x_3 - x_4). \qquad \text{restore } x_1, x_2
\end{aligned}$$

Because we started with a quartic expression, two iterations were required to reduce all the non-quadratic positive terms. This gave us two auxiliary variables in the final quadratic expression.

If you are curious about the tradeoffs for the different degree reduction methods, take a look at the review by Nike Dattani (see above footnote)!

9.3 Converting a 3-SAT decision problem to a Max-SAT optimization problem

3-SAT, which is NP complete, can only be solved via Grover's algorithm in $O(\sqrt{N})$ quantum operations where $N = 2^n$ and n is the number of input Boolean variables. Is there any advantage to converting a 3-SAT decision problem to an optimization problem and using a quantum adiabatic or variational method to solve it? Possibly. Grover's algorithm has been proven to be optimal so it is not really possible to improve upon it, but it could be matched using an adiabatic method. An adiabatic method could have advantages over the Grover type algorithm that depend upon the flexibility and accuracy of the quantum computer.

Consider a 3-SAT problem in normal conjunctive form

$$\Phi(\mathbf{x}) = h_1 \wedge h_2 \wedge h_3 .... \wedge h_k \tag{139}$$

where h_i are logical expressions (clauses) that each involve three variables; for example $x_1 \vee \bar{x}_2 \vee x_3$. The function $\Phi(\mathbf{x})$ returns 0 or 1.

How would we convert this 3-SAT problem to an optimization problem? We need to create a cost function with minima (possibly more than 1 of them) that are solutions to the 3-SAT problem and have $\Phi(\mathbf{x}) = 1$.

We construct an optimization function that counts the number of clauses that are true;

$$h_\Phi(\mathbf{x}) = h_1 + h_2 + h_3 + ... + h_k. \tag{140}$$

Here $h_\Phi(\mathbf{x})$ returns a non-negative integer and each clause is either 0 or 1.

Using the fact that $x_i \vee x_j \vee x_k = \overline{\bar{x}_i \wedge \bar{x}_j \wedge \bar{x}_k}$, we convert each clause to a Boolean cubic function

$$\begin{aligned}(x_i \vee x_j \vee x_k) \rightarrow & (1-x_i)(1-x_j)(1-x_k) \\ & = 1 - x_i - x_j - x_k + x_i x_j + x_i x_k + x_j x_k - x_i x_j x_k.\end{aligned} \tag{141}$$

If all of the variables are 0, then the resulting function on the right gives 1. If one or more of the variables is 1, then the function on the right gives 0. Thus the function on the right is a minimum when the clause on the left is true. Note that the clause on the left is Boolean and the function on the right only returns 0 or 1 so can be considered Boolean. However, the resulting function can be incorporated into a cubic optimization function within $h_\Phi(\mathbf{x})$ which returns a non-negative integer. Nowhere on the right side in equation 141 need we assume that arithmetic is mod 2.

It is straightforward to similarly convert clauses that have negated variables, for example,

$$\begin{aligned}(x_i \vee x_j \vee \bar{x}_k) \rightarrow & (1-x_i)(1-x_j)(1-\bar{x}_k) \\ & = 1 - x_i - x_j - x_k + x_i x_j + x_i \bar{x}_k + x_j \bar{x}_k - x_i x_j \bar{x}_k.\end{aligned} \tag{142}$$

Using the procedure shown in equation 141, the function $h_\Phi(\mathbf{x})$ becomes a cubic Boolean function that counts the number of satisfied clauses. If Φ has k conjunctions and the associated 3-SAT problem has a solution $\mathbf{x}^*$, (giving $\Phi(\mathbf{x}^*) = 1$) then $h_\Phi(\mathbf{x}^*) = k$. The natural number k is the maximum possible value for $h_\Phi(\mathbf{x})$.

To find the maximum possible number of satisfied clauses via minimization we use a cost function

$$C(\mathbf{x}) = -h_\Phi(\mathbf{x}) \tag{143}$$

which has minima corresponding to maxima of the clause counting function.

The cubic Boolean optimization function can be converted into a quadratic one using the procedures (Freedman/Ishikawa methods) we introduced in section 9.2.

In summary, any 3-SAT problem can be converted into a MAX-SAT Boolean optimization problem which can be expressed via a quadratic optimization function. As discussed in section 9.1, the minimum of a quadratic Boolean optimization function (QUBO) can be found on a quantum computer by finding the ground state of the associated Ising model.

10 Quantum simulation problems

10.1 Fermions as qubits via the Jordan-Wigner transformation

A system of fermions can be described by a Hamiltonian that associates energies describing interactions between fermions using a set of operators that satisfies the above commutation relations. An example is BCS theory of superconductivity.

A fermionic Hamiltonian can be converted into a spin Hamiltonian which can be implemented on a quantum computer comprised of qubits.

First we describe operators relevant for fermions. In many body problems, particles are described with creation and annihilation operators $a_i, a_i^\dagger$. Here the index labels the single particle states. For fermions, these operators satisfy anti-commutator relations

$$\{a_i, a_j^\dagger\} = a_i a_j^\dagger + a_j^\dagger a_i = \delta_{ij}\hat{I} \tag{144}$$

$$\{a_i^\dagger, a_j^\dagger\} = \{a_i, a_j\} = 0, \tag{145}$$

where $\hat{I}$ is the identity operator and the relations hold for all i, j. The second of these relations implies that $(a^i)^2 = 0$.

The operator

$$N_j = a_j^\dagger a_j$$

is Hermitian and has eigenvalues 0 or 1. The operator a_j acts like a lowering operator and $a_j^\dagger$ acts like a raising operator. The set of operators $a_j^\dagger a_j$ for different j values all commute with each other. We can choose a basis where each basis element is defined via a binary string and each digit in the tring gives the eigenvalue of $N_j = a_j^\dagger a_j$. This gives a basis of 2^n orthogonal states where n is the number of particles. If we describe each state based on how it created from a vacuum state with eigenvalues 0 for all operators, then the sign of this state depends upon the order that the raising operators are applied. Let s be a binary string comprised of digits $s_1 s_2 s_3 .....s_n$ for an n particle system and where $s_i \in \{0, 1\}$ are binary digits. The we can define a state

$$|s\rangle = (a_1^\dagger)^{s_1}(a_2^\dagger)^{s_2}....(a_n^\dagger)^{s_n} \left|\tilde{0}\right\rangle \tag{146}$$

where $\left|\tilde{0}\right\rangle$ is the ground state that has $N_j \left|\tilde{0}\right\rangle = 0$ for all j. There are 2^n of these states, and that means the space is equivalent to an n qubit system. However, it is inconvenient to use the operators $a_j, a_j^\dagger$ because they don't commute.

The Jordan-Wigner transform is based on the following operators

$$Z_j = a_j a_j^\dagger - a_j^\dagger a_j \tag{147}$$

$$X_j = -(Z_1 Z_2 .... Z_{j-1})(a_j + a_j^\dagger) \tag{148}$$

$$Y_j = i(Z_1 Z_2 .... Z_{j-1})(a_j - a_j^\dagger). \tag{149}$$

These operators serve as Pauli matrices. While X_j, Y_j don't commute, X_j, Y_k for $j \neq k$ do commute. Their commutators are the same as those of Pauli matrices! A Hamiltonian written in terms of $a_j, a_j^\dagger$ can be converted into a form that has only Pauli matrices in it.

In terms of Pauli matrices, the inverse transform is

$$a_j = -(Z_1 .... Z_{j-1}) \otimes |0\rangle_j \langle 1|_j = -(Z_1 .... Z_{j-1}) \otimes (X_j - iY_j) \tag{150}$$

$$a_j^\dagger = -(Z_1 .... Z_{j-1}) \otimes |1\rangle_j \langle 0|_j = -(Z_1 .... Z_{j-1}) \otimes X_j + iY_j). \tag{151}$$

A system of fermions can be written in terms of a Hamiltonian that contains $a_j, a_j^\dagger$. An orthonormal basis is created using equation 146 and looks just like that of n qubits. The Hamiltonian can be transferred into one only containing Pauli matrices using the Jordan-Wigner transform given in equation 149. Then the entire system can be simulated on a quantum computer comprised of qubits.

A disadvantage of the Jordan-Wigner transform is that it is 'non-local'. What is meant is that for j larger, a_j and $a_j^\dagger$ depend on a large number of Pauli Z matrices and so require operations on a large number of qubits. However, for spin chains and Hamiltonians with quadratic interactions, apparently the resulting Hamiltonian in terms of generalized Pauli operators is not all that complicated.

11 Variational Quantum Methods

Variational Quantum Methods are a strategy for performing significant computations on NISQ devices.

Consider a Hamiltonian H (a Hermitian operator) with eigenvectors $|v_j\rangle$ and eigenvalues λ_j. The Hamiltonian can be written as

$$H = \sum_j \lambda_j |v_j\rangle \langle v_j|$$

The expectation of H for any state vector

$$\langle \psi | H | \psi \rangle = \langle \psi | \sum_j \lambda_j |v_j\rangle \langle v_j | \psi \rangle \tag{152}$$

$$= \sum_j \lambda_j | \langle \psi | v_j \rangle |^2 \tag{153}$$

Notice that $|\langle\psi|v_j\rangle|^2 \geq 0$ so this is a sum of eigenvalues where each value is weighted by a positive number.

The minimum possible value for $\langle\psi| H |\psi\rangle$ is equal to the smallest eigenvalue of H.

By iterating on the choice of $|\psi\rangle$, it is possible to estimate the ground state energy of a Hamiltonian.

What is being varied? The eigenfunction $|\psi\rangle$.

What is being minimized? $\langle\psi| H |\psi\rangle$.

What is the goal? To estimate the ground state energy of H and find the associated ground state eigenvector. Apparently the resulting ground state energy has much better accuracy (smaller error) than the resulting ground state wavefunction.

11.1 The Variational Quantum Eigensolver (VQE)

Relevant references:

The theory of variational hybrid quantum-classical algorithms, McClean+15, `https://arxiv.org/pdf/1509.04279.pdf`

A cross-disciplinary introduction to quantum annealing-based algorithms, Venegas-Andraca et al., Contemporary Physics Vol. 59, Issue 02, pp. 174-196 (2018), `https://arxiv.org/abs/1803.03372`

To carry out a variational algorithm a method for varying the ansatz eigenvector is required. On a quantum computer you can generate $|\psi\rangle$ using a unitary operation $U(\boldsymbol{\theta})$ that is a function of a vector of angles (which can be varied) and a simple base state $|\psi_0\rangle$ with

$$|\psi\rangle = U(\boldsymbol{\theta}) |\psi_0\rangle .$$

The base state $|\psi_0\rangle$ might be something like $|0\rangle$ or $H^n |0\rangle$ in a n qubit system. The vector $\boldsymbol{\theta}$ of angles allow you to vary the choice of $|\psi\rangle$.

Measurement can be done of H of itself giving $\langle\psi(\boldsymbol{\theta})| H |\psi(\boldsymbol{\theta})\rangle$. A classical non-linear optimizer is used to determine a new value for $\boldsymbol{\theta}$ that will decrease the value for $\langle\psi(\boldsymbol{\theta})| H |\psi(\boldsymbol{\theta})\rangle$. The process is repeated until a convergence test is satisfied.

How can measurement of $\langle H\rangle$ be done? You can use phase estimation and an algorithm like the Grover algorithm. Or, on a quantum computer, any Hermitian operator can be written as a sum of Pauli operators; $H = \sum_i h_i E_i$ where E_i is a product of Pauli operators. As $\langle H\rangle = \sum_i h_i \langle E_i\rangle$ the expectation of H could be computed using measurements of products of Pauli operators.

12 Next!

What to discuss next? Using quantum computers to carry out simulations of quantum systems? Fermion raising and lower ops? Accuracy of Trotterization and ways to improve this with local Hamiltonians?

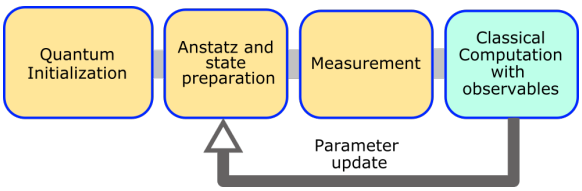

Figure 20: A flow chart for a VQE algorithm. A classical non-linear optimizer is used to minimize the expectation value by varying ansatz parameters.