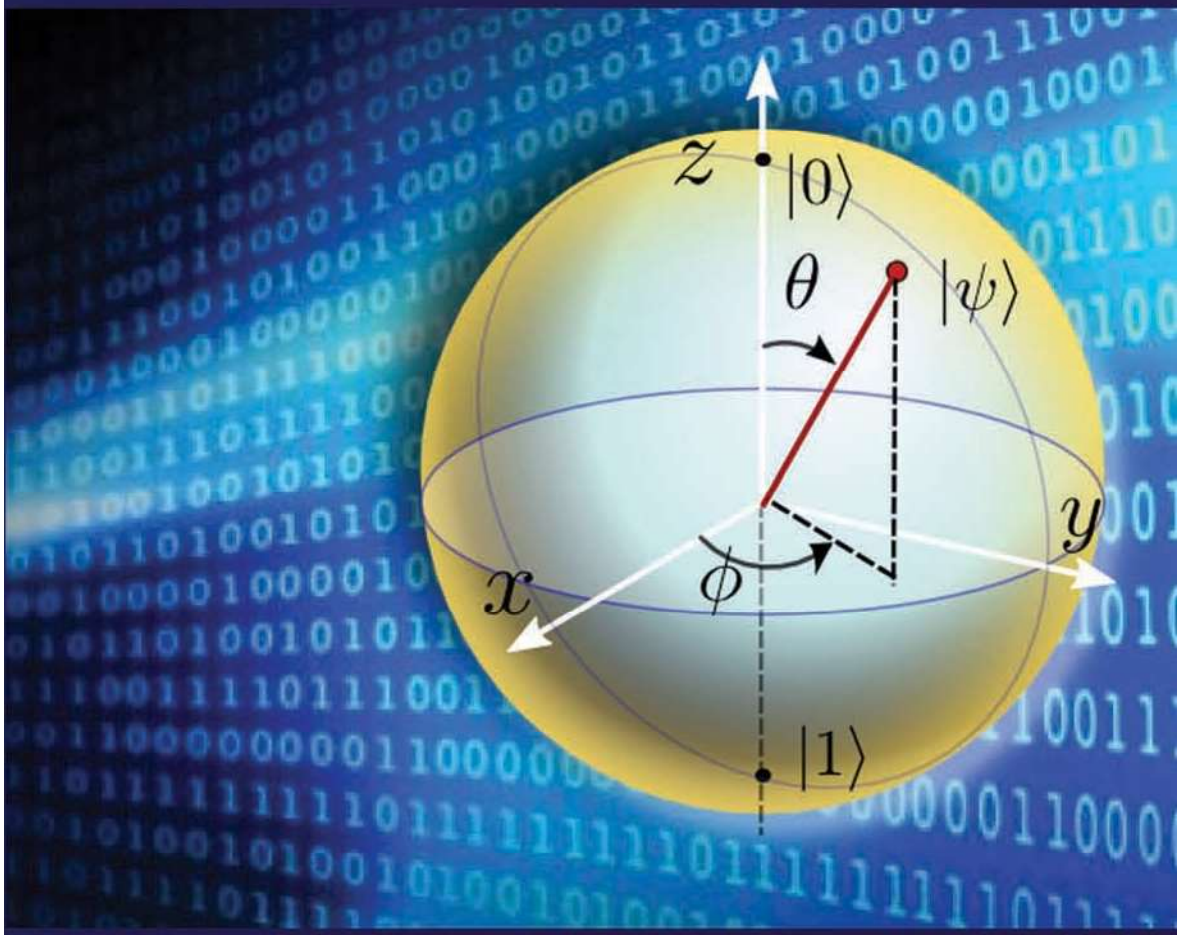
INTRODUCTION TO QUANTUM PHYSICS AND INFORMATION PROCESSING



Radhika Vathsan

CRC Press
Taylor & Francis Group

INTRODUCTION TO QUANTUM PHYSICS AND INFORMATION PROCESSING

Chapter 8

Quantum Algorithms

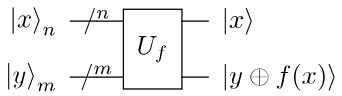
In the last chapter, we introduced the circuit model for quantum computation, where a computation is essentially the evaluation of a function. In the binary system of computation a function is a map $f : \{0,1\}^n \mapsto \{0,1\}^m$. The function takes an n-bit input and produces an m-bit output. However, this can be regarded as m functions $f_i : \{0,1\}^n \mapsto \{0,1\}$, that take an n-bit input and give a one-bit output, each of which is one bit of $f(x)$. We can thus restrict our attention to $n \to 1$ functions alone, in other words, we can reduce our problem to questions with yes/no answers, a so-called binary decision problem.

In order to evaluate a function, we have to feed it all allowed inputs and tabulate the corresponding outputs. We can construct circuits evaluating different functions by suitable combinations of gates. Now a given function evaluation or computational task is to be optimized by exploiting quantum mechanics. How is this done efficiently?

We have seen that the quantum function evaluator for $f : \{0,1\}^n \mapsto \{0,1\}^m$ is defined by the 2^{n+m}-square unitary operator of Definition 7.1:

$$U|x\rangle_n|y\rangle_m = |x\rangle_n|y \oplus f(x)\rangle_m.$$

This is essentially an **f-controlled NOT gate** as in Figure 7.15:



For $y = 0$ the output is simply $|f(x)\rangle$. If the function evaluator is fed a uniform superposition of all n-qubit basis states, then the linearity of the operator U_f ensures that the output is a uniform superposition of functions on each input, (entangled with the corresponding input state,) as in Figure 8.1.

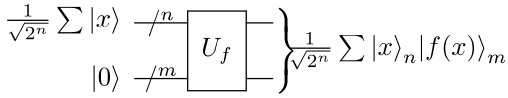


FIGURE 8.1: The quantum function evaluator with a uniform superposition.

Here the top line is sometimes called the *input register*, since the input to the function is fed through it, and the bottom one is the so-called *output register*, since it reflects the state containing the evaluated function.

The advantage of the quantum function evaluator is that it can take all possible inputs simultaneously as a superposition of states, and the corresponding outputs are all simultaneously present in the output state. This has often been called quantum parallelism. However, in this basic form it gives us no advantage, since to actually discover the value of the function, we must measure the output, upon which the output state will collapse to *one* of the possible outputs at random. The trick to making quantum computing work is to cleverly manipulate this basic function evaluator in such a way that the probability amplitude for the answer to the problem is maximum. It is quantum interference that enables this to happen. If this had not been possible, quantum computing would have been a forgotten chapter in the history of science. As it happens, this field received new impetus when Peter Shor shook up the world in 1994 with his famous algorithm for finding the prime factors of large integers.

All known quantum algorithms seem to fall into three broad classes:

1. Based on the Fourier transform: Deutsch–Josza, Shor's algorithm etc.
2. Based on quantum search, involving amplitude amplification: Grover's algorithm etc.
3. Quantum simulations.

In this chapter we will examine the first two kinds, leaving the last to more physics-specific texts. The algorithms are typically framed as yes-no answers to inputs to the function evaluator treated as a black box (Figure 8.2). This is also referred to as querying the oracle, as the unknown function evaluator is regarded, like a mysterious priestess who will only give single-bit answers when questioned!

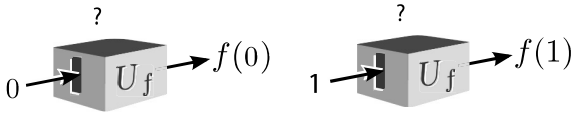


FIGURE 8.2: Classical black box function evaluator as an oracle.

8.1 The Deutsch Algorithm

Let's start with 1-bit functions $f : \{0,1\} \mapsto \{0,1\}$. There are totally four possible functions, and evaluated on inputs 0 and 1 can give answers 0 or 1. To actually determine which of these our black box is we need to query it with both inputs, whether classically or otherwise, and we obtain no

advantage using quantum computing. However, as David Deutsch [24] showed in 1985, it is possible to distinguish the function on the basis of some property, more efficiently in the quantum case. The particular classification Deutsch's algorithm considers is the following: they are either *constant* ($\mathbb{C}$), i.e., $f(0) = f(1)$, or *balanced* ($\mathbb{B}$), i.e., the outputs contain an equal number of 0's and 1's ($f(0) = \overline{f(1)}$).

Example 8.1.1. For $n > 1$, functions need not fall into the classes $\mathbb{C}$ or $\mathbb{B}$ alone. For example, consider f_1 and f_2 defined by:

$$f_1: \begin{array}{ll} f(00) = & 0 \\ f(01) = & 1 \\ f(10) = & 0 \\ f(11) = & 1 \end{array}, \quad f_2: \begin{array}{ll} f(00) = & 0 \\ f(01) = & 1 \\ f(10) = & 0 \\ f(11) = & 0 \end{array}$$

Here, f_1 is balanced while f_2 is neither constant nor balanced.

Deutsch's algorithm[1] is formulated for the following problem. Although it might seem contrived, it is the first algorithm to demonstrate the principles of the new paradigm.

The problem: given a black-box (oracle) that implements a 1-bit function $f(x)$, how will you determine whether the function belongs to class $\mathbb{C}$ or to class $\mathbb{B}$ with a minimum number of runs of the black box (or equivalently, queries to the oracle)?

Classically, it is clear that we have to run the machine twice, with inputs 0 and 1.

The Deutsch algorithm shows how this problem can be solved in just *one* run of the black box. The circuit is shown in Figure 8.3, that we will work through step by step.

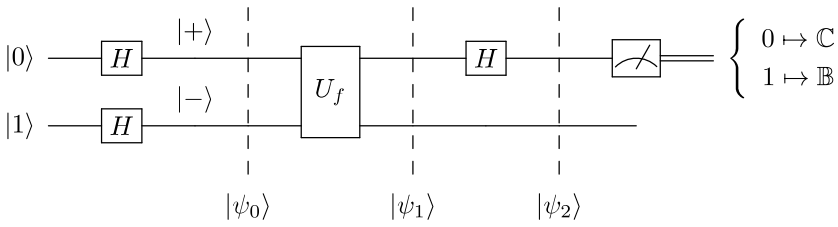


FIGURE 8.3: The Deutsch algorithm.

Step 1: Supply as input the uniform superposition

$$H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle). \tag{8.1}$$

[1]The presentation given here is not the original one in [24] but an improved version presented first by [19].

Step 2: On the bottom register, supply the state $H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$. This is the crucial feature that introduces useful interference in the result. The reason for this will be clear when we evaluate the output of the black box. So the input state is

$$\begin{aligned} |\psi_0\rangle &= \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \\ &= \frac{1}{2}[|00\rangle - |01\rangle + |10\rangle - |11\rangle] \end{aligned} \tag{8.2}$$

Step 3: Run the function evaluator. The output is

$$\begin{aligned} |\psi_1\rangle &= U_f|\psi_0\rangle \\ &= \frac{1}{2}\left[|0\rangle|f(0)\rangle - |0\rangle|\overline{f(0)}\rangle + |1\rangle|f(1)\rangle - |1\rangle|\overline{f(1)}\rangle\right] \\ &= \frac{1}{2}|0\rangle\left[|f(0)\rangle - |\overline{f(0)}\rangle\right] + \frac{1}{2}|1\rangle\left[|f(1)\rangle - |\overline{f(1)}\rangle\right]. \end{aligned} \tag{8.3}$$

Step 4: Measure the top register in the X-basis. That is, change basis by applying the H gate on the first qubit and then measure it. Just before the measurement, the output state on both wires is

$$\begin{aligned} |\psi_2\rangle &= H_1|\psi_1\rangle \\ &= \frac{1}{2\sqrt{2}}|0\rangle\left[|f(0)\rangle - |\overline{f(0)}\rangle + |f(1)\rangle - |\overline{f(1)}\rangle\right] \\ &\quad + \frac{1}{2\sqrt{2}}|1\rangle\left[|f(0)\rangle - |\overline{f(0)}\rangle - |f(1)\rangle + |\overline{f(1)}\rangle\right] \end{aligned} \tag{8.4}$$

If the function is $\mathbb{C}$, then $f(0) = f(1)$ and the amplitude for $|0\rangle$ is 1 while that for $|1\rangle$ is 0. On the other hand, when f is $\mathbb{B}$ then $f(0) = \overline{f(1)}$ and the amplitude for $|1\rangle$ is 1 while that for $|0\rangle$is 0. Thus a measurement of the output gives us the answer to the query with certainty. We have run the function evaluator only once. The quantum advantage has given us a double speedup in this case.

The reason why this works is that Step 2 implements the so called "*phase kickback*" trick. If the state $|-\rangle$ on the lower register fed into the black box, then the output acquires a phase that depends on $f(x)$. This phase can effectively be regarded as attached to the state of the upper register.

$$\begin{aligned} U_f\left[|x\rangle \otimes \frac{(|0\rangle - |1\rangle)}{\sqrt{2}}\right] &= \frac{1}{\sqrt{2}}|x\rangle\left[|f(x)\rangle - |\overline{f(x)}\rangle\right] \\ &= \begin{cases} \frac{1}{\sqrt{2}}|x\rangle(|0\rangle + |1\rangle) & \text{if } f(x) = 0, \\ \frac{1}{\sqrt{2}}|x\rangle(|1\rangle - |0\rangle) & \text{if } f(x) = 1 \end{cases} \\ &= (-1)^{f(x)}|x\rangle\frac{1}{\sqrt{2}}\left[|0\rangle - |1\rangle\right]. \end{aligned} \tag{8.5}$$

The output is separable, with the lower register unchanged in state $|-\rangle$, while the upper register is effectively the input with an $f(x)$-dependent phase.

With this effect, we can re-analyze the algorithm with the uniform superposition $|x\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ in the input register:

$$\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \xrightarrow{U_f} \frac{1}{\sqrt{2}}\left[(-1)^{f(0)}|0\rangle + (-1)^{f(1)}|1\rangle\right] \tag{8.6}$$

$$\xrightarrow{H} \frac{1}{\sqrt{2}}\left[\left((-1)^{f(0)} + (-1)^{f(1)}\right)|0\rangle + \left((-1)^{f(0)} - (-1)^{f(1)}\right)|1\rangle\right] \tag{8.7}$$

where it's obvious that a measurement gives $|0\rangle$ if $f(x)$ is $\mathbb{C}$ and $|1\rangle$ if $f(x)$ is $\mathbb{B}$.

8.1.1 Deutsch–Josza algorithm

The Deutsch algorithm was extended to n-bit functions by Josza and others in 1992 [26].

The problem: Given an $n \to 1$ function $f : \{0,1\}^n \mapsto \{0,1\}$ that is guaranteed to be either constant or balanced, find out which it is in a minimum number of runs.

Classically, we would proceed by querying the oracle with each n-bit number. If we find an answer that is not equal to the previous one then we have a balanced function. In worst-case scenario, we might find the same $f(x)$ until the half the possible inputs, i.e., after querying the function $2^n/2$ times. The answer to the next query would solve the problem. Thus we need to run the oracle at worst $2^{n-1}+1$ times: exponential in the number of bits of input.

The quantum algorithm achieves the distinction in just one run! This is a dramatic speedup indeed. The circuit (Figure 8.4) is an n-qubit extension of that for the Deutsch problem:

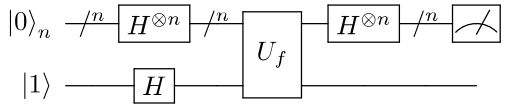


FIGURE 8.4: The circuit for the Deutsch–Josza algorithm.

The input to the circuit is the uniform n-qubit superposition

$$H^{\otimes n}|0\rangle_n = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle. \tag{8.8}$$

Due to the phase-kickback trick, the output of U_f on the input register is the

superposition

$$|\psi_1\rangle = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} (-1)^{f(x)} |x\rangle. \tag{8.9}$$

After the Hadamard, this state becomes

$$H^{\otimes n}|\psi_1\rangle = \frac{1}{2^n} \sum_{x,y=0}^{2^n-1} (-1)^{[f(x)+x\cdot y]} |y\rangle. \tag{8.10}$$

Here, we have used the result of Equation 7.28 and $x \cdot y$ is the bitwise product of x and y summed modulo 2, as in Equation 7.29. Now when $f(x)$ is constant, the amplitude for the state $|0\rangle$ in this superposition is

$$\text{coefficient of } |0\rangle = \frac{1}{2^n} \sum_x (-1)^{f(x)} = 1.$$

In other words the probability of getting $|0\rangle$ is one for a constant function. Whereas if $f(x)$ is balanced, then the amplitude for $|0\rangle$ is a sum of an equal number of $+1$s and -1s, that is, zero. Thus if the function is balanced, the output measures to any number *other* than 0. We thus distinguish the two classes in one run of the black box, which is nearly an n-fold speedup compared to the classical case.

8.2 The Bernstein–Vazirani Algorithm

We'll now look at algorithms that show more substantial speedups compared to classical ones. One such algorithm was invented by Umesh Vazirani and his student Ethan Bernstein in 1993 [11]. This algorithm identifies a linear Boolean function in one query of the oracle.

The problem: given a function evaluator for

$$f : \{0,1\}^n \mapsto \{0,1\} \text{ where } f(x) = a \cdot x, \quad a \in [0, 2^n], \tag{8.11}$$

and the dot is a bitwise product with modulo 2 addition:

$$a \cdot x \equiv a_0 x_0 \oplus a_1 x_1 \oplus \cdots \oplus a_{n-1} x_{n-1}, \tag{8.12}$$

determine the function, or in other words find a.

Example 8.2.1. An example of such a function for $n = 2$ and $a = 11$, which evaluates to

$$f(00) \quad = \quad 0$$

$$\begin{aligned} f(01) &= 0.1 \oplus 1.1 = 1 \\ f(10) &= 1.1 \oplus 0.1 = 1 \\ f(11) &= 1.1 \oplus 1.1 = 0 \end{aligned}$$

Classically, we can determine the k^{th} bit of a if we feed the oracle the input $x = 2^k$, that has only the k^{th} bit as 1 and all the rest as 0. This becomes obvious when you look at the binary expansion of a:

$$a = a_0 + a_1 2^1 + \cdots + a_k 2^k + \ldots \implies a_k = a \cdot 2^k. \tag{8.13}$$

This calls the function n times.

The quantum algorithm, which uses the same circuit as for the Deutsch–Josza algorithm, succeeds with *one* call!

Let's analyze the output of the circuit of Figure 8.4 for this form of the function:

$$\sum_x \sum_y \frac{1}{2^n}(-1)^{f(x)+x\cdot y}|y\rangle \otimes |-\rangle \quad = \quad \frac{1}{2^n}\sum_y \left[\sum_x (-1)^{a\cdot x + y\cdot x}\right] |y\rangle \otimes |-\rangle \quad . \tag{8.14}$$

The amplitude for $|y\rangle$ is $\frac{1}{2^n}\sum_x (-1)^{a\cdot x+y\cdot x} = \frac{1}{2^n}\sum_x (-1)^{(a+y)\cdot x} = 1$ if $y = a$! It's easy to see why it is zero for all other values of y. Thus with certainty, the output of the circuit gives us a.

A more explicit way of seeing why this works is by analyzing the circuit for U_f. This analysis is lucidly given in Mermin [48]. The black box for $a\cdot x$ flips the bit in the lower register whenever a bit of the input x and the corresponding bit of a are both 1. For instance, suppose we had $a = 11010$ with $n = 5$. Then it can easily be seen that $a \cdot x$ is implemented by the circuit of Figure 8.5.

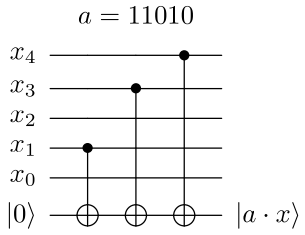


FIGURE 8.5: A circuit that executes U_f for $f = 11010 \cdot x$.

Coming to the circuit for solving the Bernstein–Vazirani problem, it has an H gate before each qubit enters the function evaluator and after. This is true even of the lower register, which can be thought of as initialized to $|1\rangle$. Note that an H gate before and after a CNOT interchanges the roles of the control and target qubits (see Figure 7.8).

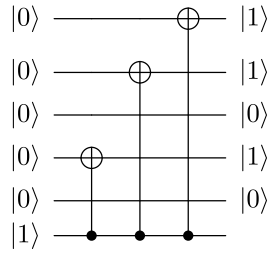


FIGURE 8.6: Analysis of circuit for the Bernstein–Vazirani algorithm for $a = 11010$.

The solution is therefore the circuit of Figure 8.6, whose output directly reads out the bits of a.

The algorithm thus gives an n-fold speedup over the classical case.

8.3 Simon's Algorithm

Even though the Bernstein–Vazirani algorithm offers such a great speedup, the classical solution is still not exponential. Daniel Simon came up with an algorithm [65] in 1994 that is the first to demonstrate a dramatic exponential speedup over a hard classical problem, but the solution is probabilistic. This feature is characteristic of many quantum algorithms. Simon's problem also illustrates a class of problems that basically use Fourier transforms, in the form of the amplitudes of the output states that "interfere" to give a large probability for the expected solution.

The problem: Given a black box implementing a function

$$f : \{0,1\}^n \mapsto \{0,1\}^{n-1} \text{ such that} f(x \oplus a) = f(x), \quad a \in [0, 2^n - 1], \quad (8.15)$$

determine a with the minimum number of queries to the box.

Example 8.3.1. The functions considered in Simon's algorithm can be thought of as "periodic" under bitwise addition. For example, let's look at the 3-bit function

x	000	001	010	011	100	101	110	111
$f(x)$	3	2	2	3	1	4	4	1

The first repetition is of the value $f(1) = f(2)$. The "period" is therefore $a = 001 \oplus 010 = 011 = 3$. You can verify that all the other repetitions also satisfy the same condition.

The classical solution to this problem is *hard*, i.e., the number of runs of the function grows exponentially as the size of the input. We would query the oracle with successive values of n-bit numbers x until we found a repeated value for the output: $f(x_i) = f(x_j)$. Then we could calculate $a = x_i \oplus x_j$. However, a could be any one of 2^n possible numbers. By the m^{th} run, $\frac{1}{2}m(m-1)$ pairs have been compared and eliminated as possible a's. For reasonable chance of success, we need $\frac{1}{2}m(m-1) \geq 2^n \implies$ a lower bound on the number of trials $m = \Omega(2^{n/2})$, which is exponential in the number of bits.

The quantum circuit (Figure 8.7) that solves this problem is essentially the same as the Deutsch–Josza circuit except that the lower register is also expanded to n qubit, and initialized to $|0\rangle_n$ (we dispense with the phase kickback).

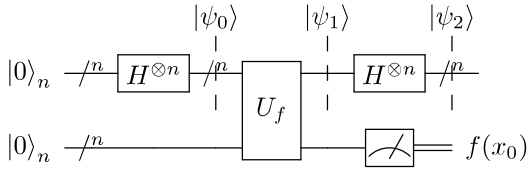


FIGURE 8.7: The circuit for the Simon algorithm.

The input to the oracle gives us

$$U_f|\psi_0\rangle \otimes |0\rangle = U_f \left[\sum_{x=0}^{2^n-1} |x\rangle \otimes |0\rangle\right] = \sum_{x=0}^{2^n-1} |x\rangle \otimes |f(x)\rangle. \tag{8.16}$$

In order to analyze the solution, let us use the reverse of the principle of delayed measurement, and assume we measure the lower register after the action of U_f. Let's denote the outcome by $f(x_0)$, which is generated from two possible inputs x_0 or $x_0 \oplus a$. The top register therefore collapses to a superposition of these two states alone:

$$|\psi_1\rangle = \frac{1}{\sqrt{2}} \left(|x_0\rangle + |x_0 \oplus a\rangle\right). \tag{8.17}$$

If we now apply H to each qubit in the upper register, we get

$$|\psi_3\rangle = \frac{1}{\sqrt{2}} \frac{1}{\sqrt{2^n}} \sum_{y=0}^{2^n-1} \left[(-1)^{x_0 \cdot y} + (-1)^{(x_0 \oplus a) \cdot y}\right] |y\rangle. \tag{8.18}$$

Now $a \cdot y$ is either 0 or 1. If $a \cdot y = 1$, then the amplitude for $|y\rangle$ is zero and all those states do not occur in the output. Thus the only states that can be measured in the output are those for which the condition $a \cdot y = 0$ is satisfied. This is a binary algebraic equation with n unknowns (the bits of a). We can find a if we can obtain n independent equations, corresponding to n different values of y. If we repeat the experiment until we have collected n distinct,

non-zero y's then we can solve for the bits of a. It is not guaranteed that we will get a distinct y on each run, so we may most probably have to run the oracle more than n times.

To determine the complexity of this problem, we will need to estimate how the number of runs of the oracle scales with n. It can be shown (see Box 8.3) that the number of times the oracle has to be queried is $n + m$ where m doesn't depend on n. This algorithm is thus a sub-exponential solution to a classically hard problem.

Box 8.1: Complexity Analysis for Simon's Algorithm

As in many quantum algorithms, the analysis of why the algorithm is computationally more efficient than the classical case involves a detailed mathematical examination of the solution. In the case of Simon's algorithm, the output after measurement is an n-bit string y such that

$$a \cdot y = a_{n-1}y_{n-1} \oplus a_{n-2}y_{n-2} \oplus \cdots \oplus a_1 y_1 \oplus a_0 y_0 = 0.$$

We need to collect at least $n-1$ such *distinct* bit-strings in order to determine the coefficients a. So we need to query the oracle at least $n-1$ times and need to find a lower bound on the probability of success.

Suppose we ran the algorithm k times and got linearly independent y's. What's the probability that the next run gives a different y? The minimum probability for this occurring is

$$\frac{2^n - 2^k}{2^n} = 1 - 2^{k-n}.$$

So the probability of getting $n-1$ independent y's is just

$$\mathcal{P} = \left(1 - \frac{1}{2^n}\right)\left(1 - \frac{1}{2^{n-1}}\right)\cdots\left(1 - \frac{1}{2}\right).$$

Now notice that $(1-s)(1-t) = 1 - (s+t) + st \geq 1 - (s+t)$. So we have

$$\begin{aligned} \mathcal{P} &\geq \left(1 - \sum_{i=1}^{n} \frac{1}{2^i}\right)\frac{1}{2} \\ &\geq \left(1 - \frac{1}{2}\right)\frac{1}{2} = \frac{1}{4}. \end{aligned}$$

This means that there is a *finite* minimum probability with which we will succeed in n runs. To ensure success, we'll have to run the algorithm a few more times, *independent* of n, so that the number of runs is still $\mathcal{O}(n)$.

The trick that makes the kind of algorithms considered so far work is that the output before measuring in the X basis has $f(x)$-dependent phases. Until

now these phases were restricted to ± 1. More general phases come about if the Fourier transform is implemented. In this section, we introduced the idea of using the H gate on the output to produce interfering amplitudes. This is just a special case of the quantum Fourier transform, as we will see in the next section.

8.4 Quantum Fourier Transform and Applications

The Fourier transform, a mathematical tool named after the 18th century French mathematician Joseph Fourier, is an invaluable tool in engineering and the sciences. No technical education is complete without a firm grasp of this technique and its uses. The simplest way to understand the Fourier transform $\mathcal{F}$ of a function $f(x)$ is to imagine the function as made up of various components that are periodic (like a sine function) with a frequency y, and $\mathcal{F}(f(x))$ as a function $\tilde{f}(y)$ measuring the amplitude of each frequency component in the function. In other words, we construct a decomposition of the function in terms of the oscillatory exponential $e^{-2\pi iyx}$, where the coefficients in that decomposition are the Fourier transform:

$$f(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} dy \; e^{2\pi iyx} \tilde{f}(y). \tag{8.19}$$

This formula is said to define the *inverse Fourier transform* of $\tilde{f}(y)$, while the Fourier transform is defined as

$$\mathcal{F}(f(x)) = \tilde{f}(y) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} dx \; e^{-2\pi iyx} f(x). \tag{8.20}$$

The factor in front of the integral captures the normalization. A function can in general have an infinite number of frequency components, and the frequencies can be distributed continuously. That's how the Fourier transform is a continuous function of the frequency y.

The two Equations 8.20 and 8.19 define a *Fourier transform pair*. The Fourier transform naturally produces complex numbers, so that $f(x)$ and $\tilde{f}(y)$ are in general complex. When we compute the Fourier transform on a digital machine, we need to discretize the integral to get the Discrete Fourier Transform (DFT).

8.4.1 The discrete Fourier transform and classical algorithm

When $f(x)$ is a discrete function over the finite range $N = 2^n$ of discrete n-bit inputs x, we can think of it as a vector with N components $\{f_0 \; f_1 \; ... \; f_{N-1}\}$. The integral over x in Equation 8.20 is then a sum over an index k with

$x \to k/N$ and the limits are restricted from 0 to $N-1$. We then get the discrete Fourier transform of order N defined as

$$\tilde{f}(y) = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{-2\pi i y k/N} f_k.$$

This is another vector with N components $\{g_0\ g_1\ ...\ g_{N-1}\}$, given by

$$g_j = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{-2\pi i j k/N} f_k. \tag{8.21}$$

These are just the coefficients of orthogonal harmonic components $e^{2\pi i j k/N}$ of the function, which can be expressed as the inverse discrete Fourier transform (IDFT):

$$f_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} e^{2\pi i j k/N} g_j. \tag{8.22}$$

We can regard the DFT as a complex matrix transformation of the vector $\{f_k\}$:

$$g_j = \sum_{k=0}^{N-1} M_{jk} f_k; \qquad f_k = \sum_{j=0}^{N-1} M_{jk}^{-1} g_k, \tag{8.23}$$

where M_{jk} are the elements of an $N \times N$ matrix M given by

$$M_{jk} = \frac{1}{\sqrt{N}} e^{2\pi i j k/N} = \frac{1}{\sqrt{N}} \omega_N^{jk}. \tag{8.24}$$

Here $\omega_N = e^{2\pi i/N}$ is the N^{th} root of unity. More explicitly,

$$\begin{pmatrix} g_0 \\ g_1 \\ \vdots \\ g_{N-1} \end{pmatrix} = \frac{1}{\sqrt{N}} \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_N & \omega_N^2 & \cdots & \omega_N^{N-1} \\ 1 & \omega_N^2 & \omega_N^4 & \cdots & \omega_N^{2(N-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \omega_N^{N-1} & \omega_N^{2(N-1)} & \cdots & \omega_N^{(N-1)^2} \end{bmatrix} \begin{pmatrix} f_0 \\ f_1 \\ \vdots \\ f_{N-1} \end{pmatrix} \tag{8.25}$$

Example 8.4.1. The simple case of $N = 2$, $\omega_2 = e^{i\pi} = -1$ and

$$\text{DFT}_2 \quad = \quad \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & \omega_2 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{8.26}$$

which is just the Walsh–Hadamard transform.

Exercise 8.1. Write out the DFT matrix for $N = 4$.

Exercise 8.2. Calculate the DFT on the N-dimensional zero-vector.

Example 8.4.2. Unitarity of the DFT:

The crucial point that allows us to extend the DFT to an operator on quantum states is that it is unitary. To prove this, we need to show that

$$M^\dagger M = \mathbb{1} \implies \sum_{l=0}^{N-1} M_{jl} M^*_{lk} \quad = \quad \delta_{jk} \tag{8.27}$$

where M_{jk} is defined by Equation 8.24.

$$\text{When } j = k: \quad \frac{1}{N} \sum_l \omega_N^{jl} \omega_N^{-lj} = \frac{1}{N} \sum_l 1 = 1; \tag{8.28}$$

When $j \neq l$, then $\sum_l \omega_N^{l(j-k)}$ is the sum of N terms of a geometric series whose first term is 1 and ratio is $\omega_N^{(j-k)}$. So we have

$$\sum_l \omega_N^{l(j-k)} = \frac{1 - \omega_N^{N(j-k)}}{1 - \omega_N^{(j-k)}} = 0 \tag{8.29}$$

Thus Equation 8.27 is proved.

Box 8.2: Classical FFT Algorithm

Computing the DFT_N of a vector involves evaluating N^2 elements of the DFT matrix, and looks like a job that scales as 2^{2n} with the number of bits $n = \log_2 N$. In implementing the DFT transform on a digital machine, one can easily optimize by exploiting the properties of the integer powers of ω_N. There are cycles among elements of DFT_N, since $\omega_N^N = 1$. So while a direct matrix multiplication of the form of Equation 8.24 would typically require $\mathcal{O}(N^2)$ basic operations, the optimized fast Fourier transform (FFT) algorithm requires $\mathcal{O}(N \log_2 N)$ operations only.

For example, consider $N = 4$; $\omega_4 = e^{2\pi i/4} = i$, $\omega_4^2 = -1$, $\omega_4^4 = 1$.

$$\mathrm{DFT}_4 = \frac{1}{2} \left(\begin{array}{cc:cc} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ \hdashline 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{array} \right) \tag{8.30}$$

Now there is a relationship between the upper and lower halves of this matrix. Look at the highlighted columns, repeated for the upper and lower halves: they form a 2×2 matrix that acts on the even index components (note the index starts at 0).

$$\frac{1}{2}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} = \frac{1}{\sqrt{2}}\mathrm{DFT}_2. \tag{8.31}$$

The part that acts on the odd index components is for the upper half

$$\begin{aligned} \frac{1}{2}\begin{pmatrix} 1 & 1 \\ i & -i \end{pmatrix} &= \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix} \times \mathrm{DFT}_2 \\ &= \frac{1}{\sqrt{2}}\mathrm{Diag}(1\ \omega_4) \times \mathrm{DFT}_2. \end{aligned} \tag{8.32}$$

The negative of this acts on the lower half. Thus the DFT of a 4-d vector is reduced to two DFT's of a 2-d vector. This is at the heart of the classical FFT algorithm.

The above example shows that DFT_N can be reduced to $\mathrm{DFT}_{N/2}$. The FFT algorithm works by recursively dividing the original vector into even numbered and odd numbered elements, until at the final stage there are just two terms and DFT_2 can be applied. The process is then reversed by successively doubling the vectors and eventually covering the entire input. Let's see how this is possible in general: let $N = 2M$.

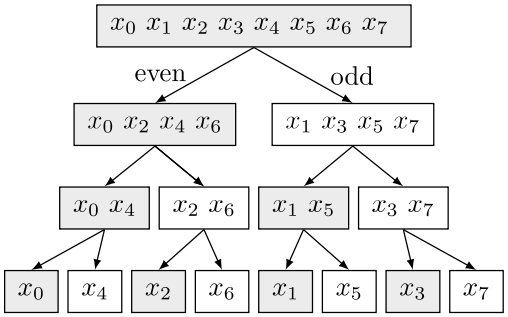
$$\mathrm{DFT}_N(f(x)) = \tilde{f}(y) = \frac{1}{\sqrt{2M}}\sum_{x=0}^{2M-1} \omega_{2M}^{xy} f(x). \tag{8.33}$$

Breaking this up into even and odd terms,

$$\begin{aligned} \mathrm{DFT}_{2M}(f(x)) =& \frac{1}{\sqrt{2M}}\left(\sum_{x=0}^{M-1} \omega_{2M}^{2xy} f(2x) + \sum_{x=0}^{M-1} \omega_{2M}^{(2x+1)y} f(2x+1)\right) \\ =& \frac{1}{\sqrt{2}}\Bigg(\underbrace{\frac{1}{\sqrt{M}}\sum_{x=0}^{M-1} \omega_M^{xy} f(2x)}_{\mathrm{DFT}_M \text{ of even terms}} + \underbrace{\frac{1}{\sqrt{M}}\sum_{x=0}^{M-1} \omega_M^{(x)y} f(2x+1)}_{\mathrm{DFT}_M \text{ of odd terms}} \times \omega_{2M}^{y}\Bigg) \end{aligned} \tag{8.34}$$

At any stage l of evaluating the DFT, one can divide the input into two to write it in terms of $\mathrm{DFT}_{l/2}$, and continue successively until one is left with DFT_2's.

Successive division of the terms in the input into two until we reach the two-term pairs is called *decimation*. The process of decimating higher-order DFT's looks like the following for $N = 8$:



We then start evaluating upward from the 2-point DFTs, successively doubling at each stage. The generic 2-point DFT looks like Figure 8.8, called a butterfly diagram for its symmetric structure. The labels on the sides represent the multiplicative factors and two lines joined at a node represent addition of the corresponding terms.

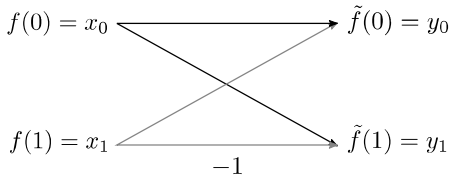


FIGURE 8.8: The 2-point DFT: butterfly diagram.

For $N = 8$, we have worked out the decimation process in Example 8.4.4. The butterfly diagram looks like Figure 8.9.

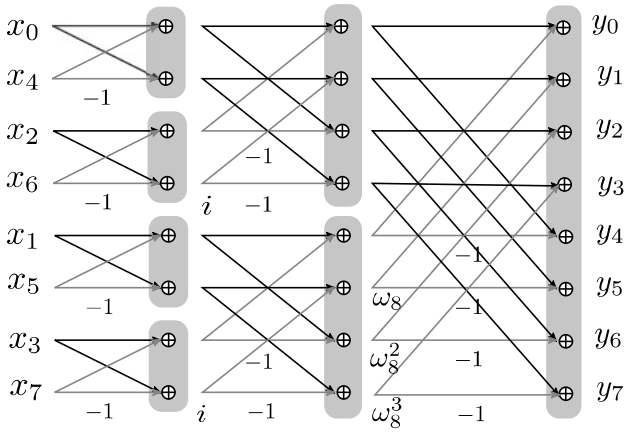


FIGURE 8.9: Butterfly diagram for computing an 8-point DFT.
The output vector $\{y_0\ y_1 \ldots\ y_8\}$ is the DFT of the input vector.

Exercise 8.3. Show how the DFT of a 6-d vector reduces to the DFT's of the even and odd indexed 3-d components.

8.4.2 Complexity of the classical FFT algorithm

Suppose the computation of DFT_N requires $T(N)$ basic operations. From Equation 8.34, we see that this is related to $T(N/2)$ since we need to evaluate two DFT's of order $N/2$ and also do N multiplications of the exponential factors. Thus,

$$T(N) = 2T(N/2) + \mathcal{O}(N). \tag{8.35}$$

Using this recursively to solve for $T(N)$ one gets

$$T(N) = \mathcal{O}(N \log_2 N). \tag{8.36}$$

8.5 Definition of the QFT from Discrete Fourier Transform

The quantum Fourier transform (QFT) is simply the DFT operation on the amplitudes of a quantum state. The DFT matrix is unitary, and can therefore represent a quantum transformation. We can define the QFT (order $N = 2^n$) of an n-qubit basis state $|x\rangle$ by

$$\hat{\mathcal{F}}_N|x\rangle = \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} \omega_N^{xy}|y\rangle. \tag{8.37}$$

Interestingly, as we have seen in Equation 8.26, the QFT transform for $n = 2$ is just the Hadamard gate.

When applied to a superposition state $|\psi\rangle = \sum_i C_i|i\rangle$, the QFT performs a DFT on the coefficients C_i:

$$\begin{aligned}
\hat{\mathcal{F}}_N|\psi\rangle &= \sum_i C_i U_N|i\rangle \\
&= \frac{1}{\sqrt{2^n}} \sum_i C_i \sum_j \omega_N^{ij}|j\rangle \qquad (8.38)\\
&= \sum_j \left(\mathrm{DFT}_N C\right)_j |j\rangle. \qquad (8.39)
\end{aligned}$$

where C denotes the vector of coefficients in the state representation.

An efficient algorithm for evaluating the QFT is inspired by the FFT algorithm, where we compute the bit-wise breakup of the action of $\hat{\mathcal{F}}$ on its

input. Remember, $|y\rangle_n = \bigotimes_{j=0}^{n-1} |y_j\rangle$. Each y_j takes a value 0 or 1. The QFT has superpositions of $|y\rangle$ with a phase $\omega_N^{xy/N}$, containing the integer product of x and y. We want to break this up into the constituent bits, the y_js. So we write

$$xy = \left(x_0 + 2x_1 + \cdots + 2^{n-1}x_{n-1}\right)\left(y_0 + 2y_1 + \cdots 2^{n-1}y_{n-1}\right). \tag{8.40}$$

Now any product in this expansion that has a coefficient of 2^n or higher can be dropped since it would contribute unity to the phase: $\omega_N^{2n} = 1$. So we find

$$\begin{aligned}
\frac{xy}{N} &= \left(\frac{x_0}{2^n} + \frac{x_1}{2^{n-1}} + \cdots + \frac{x_{n-1}}{2}\right) y_0 \\
&+ \left(\frac{x_0}{2^{n-1}} + \frac{x_1}{2^{n-2}} + \cdots + \frac{x_{n-2}}{2}\right) y_1 \\
&\vdots \\
&+ \left(\frac{x_0}{2^2} + \frac{x_1}{2}\right) y_{n-2} \\
&+ \left(\frac{x_0}{2}\right) y_{n-1}.
\end{aligned} \tag{8.41}$$

Using the binary "point" notation

$$0.x_1x_2x_3...x_n = \frac{x_1}{2} + \frac{x_2}{2^2} + \frac{x_3}{2^3} + ...\frac{x_n}{2^n}, \tag{8.42}$$

$$\frac{xy}{N} = y_0(0.x_{n-1}x_{n-2}\cdots x_0) + y_1(0.x_{n-2}\cdots x_0) + \cdots + y_{n-1}(0.x_0). \tag{8.43}$$

Using this to write the QFT in bit-wise expansion, we can associate an exponential factor with each bit of y, the output becomes the following product state:

$$\begin{aligned}
\hat{\mathcal{F}}_N|x\rangle &= \frac{1}{\sqrt{2^n}} \sum_y e^{2\pi i xy/N} |y_0\rangle \otimes |y_1\rangle \cdots |y_{n-1}\rangle \\
&= \frac{1}{\sqrt{2^n}} \left(\sum_{y_0=0,1} e^{2\pi i y_0 (0.x_{n-1}x_{n-2}\cdots x_0)} |y_0\rangle \right) \\
&\quad \otimes \left(\sum_{y_1=0,1} e^{2\pi i y_1 (0.x_{n-2}\cdots x_0)} |y_1\rangle \right) \otimes \\
&\quad \cdots \otimes \left(\sum_{y_{n-1}=0,1} e^{2\pi i y_{n-1} (0.x_0)} |y_{n-1}\rangle \right) \\
&= \left(\frac{|0\rangle + e^{2\pi i (0.x_{n-1}x_{n-2}\cdots x_0)}|1\rangle}{\sqrt{2}} \right) \otimes \left(\frac{|0\rangle + e^{2\pi i (0.x_{n-2}\cdots x_0)}|1\rangle}{\sqrt{2}} \right) \otimes \\
&\quad \cdots \otimes \left(\frac{|0\rangle + e^{2\pi i (0.x_0)}|1\rangle}{\sqrt{2}} \right)
\end{aligned} \tag{8.44}$$

Each term in the product of Equation 8.44 is the state of an output qubit for the corresponding qubit of the input. This translates into a circuit for evaluating $\hat{\mathcal{F}}_{2^n}$. The order of occurrence of the terms must be noted: the first term is the least significant bit, and the last is the most significant bit of the output.

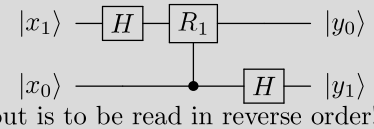
Example 8.5.1. QFT circuit for $n = 2$:

$$|x_1 x_0\rangle \xrightarrow{\hat{\mathcal{F}}} \frac{|0\rangle + e^{2\pi i(0.x_0)}|1\rangle}{\sqrt{2}} \otimes \frac{|0\rangle + e^{2\pi i(0.x_1 x_0)}|1\rangle}{\sqrt{2}}$$

This means

$$\begin{aligned} |x_1\rangle &\longrightarrow \frac{|0\rangle + e^{2\pi i(\frac{x_0}{2})}|1\rangle}{\sqrt{2}} = |y_1\rangle \\ |x_0\rangle &\longrightarrow \frac{|0\rangle + e^{2\pi i(\frac{x_1}{2}+\frac{x_0}{4})}|1\rangle}{\sqrt{2}} = |y_0\rangle \end{aligned}$$

Here $|y_1\rangle$ has an x_0-dependent phase of $e^{i\pi}$ for $|1\rangle$, which can be obtained by an H acting on $|x_0\rangle$. Similarly $|y_0\rangle$ has a x_1-dependent phase of $e^{i\pi}$ and an x_0-dependent phase of $e^{i\pi/2}$ for $|1\rangle$. The first is obtained by an H on $|x_1\rangle$ while the second is the x_0-controlled action of the gate $R_1 = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/2} \end{pmatrix}$:



Note that the output is to be read in reverse order!

Exercise 8.4. Work out the circuit for the QFT for $n = 3$.

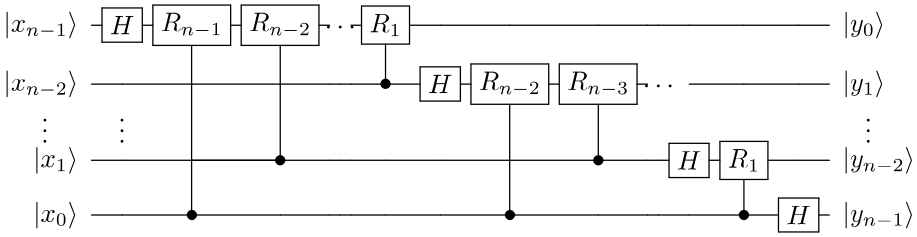


FIGURE 8.10: Circuit for the quantum Fourier transform $\hat{\mathcal{F}}_{2^n}$, on n qubits.

You should now be able to work out that Figure 8.10 is an efficient quantum circuit for the QFT on n qubits.. We require controlled phase gates, with phase

matrices like

$$R_d = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/2^d} \end{pmatrix}, \tag{8.45}$$

where d can be interpreted as the distance from the control bit. Notice that the output bits are in reverse order. One can either agree to read the output in reverse order or to perform a swap at the end.

The efficiency of this circuit is related to the number of basic gate operations required per input bit. We can easily see that this is n H-gates and $n(n-1)/2$ C-R-gates for n bits, which is $\mathcal{O}(n^2)$. This is *exponentially* faster than the classical FFT which takes $\mathcal{O}(n2^n)$. Hurray for quantum algorithms!

But before we exult too much, observe that the output of the quantum Fourier transform is a superposition of basis states whose phases represent the Fourier transform of the corresponding input bit. A measurement at the end of the above circuit gives us **no** information whatsoever about the Fourier transform of the input! So we cannot use this circuit as a super-efficient Fourier transform computer! Instead, we have to incorporate it in procedures that require FT-dependent phases. And Peter Shor did just that in his path-breaking algorithm for prime factorization.

8.5.1 Period-finding using QFT

Preliminary to the Shor algorithm, let's focus on one that lends itself naturally to the QFT: computing the period r of a periodic n-bit function

$$f : \{0,1\}^n \mapsto \{0,1\}^n \text{ such that } f(x+r) = f(x), \quad r \in [0, 2^n - 1]. \tag{8.46}$$

We will take $2^n = N$ in what follows. The function could repeat more than once in the interval $[0, N-1]$, so we have

$$f(x + kr) = f(x), \qquad kr < N.$$

We assume we are presented with a black box (oracle) that evaluates such a function. The algorithm uses a circuit that is a direct extension of Simon's algorithm (Section 8.3), in which we'll use the full QFT instead of the 1-bit version (the Hadamard transform) used there. The circuit (Figure 8.11) is straightforward.

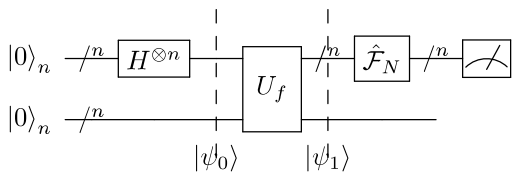


FIGURE 8.11: Circuit for quantum period finding.

The input to the U_f black box is once again

$$|\psi_0\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle_n \otimes |0\rangle_n, \tag{8.47}$$

So the output ought to be

$$|\psi_1\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle \otimes |f(x)\rangle. \tag{8.48}$$

We will again assume that we measure the lower register at this point, obtaining some number f_0. Then the top register collapses to a superposition of only those states $|x\rangle$ for which $f(x) = f_0$. All such x's are of the form $x_0 + kr$ for some $x_0 < r$, and some integer $k : kr < N$. Suppose the number of periods within the interval $[0, N-1]$ is p:

$$p = [N/r]\,, \tag{8.49}$$

where the square bracket notation stands for the ceiling function (greatest integer less than the argument). The state of the computer is then a superposition of p terms of the form

$$|\psi_1\rangle = \frac{1}{\sqrt{p}} \sum_{k=0}^{p-1} |x_0 + kr\rangle \otimes |f_0\rangle. \tag{8.50}$$

Now subjecting the top register to a QFT, we get

$$\hat{\mathcal{F}}_N \left(\frac{1}{\sqrt{p}} \sum_{k=0}^{p-1} |x_0 + kr\rangle \right) = \sum_{y=0}^{N-1} \left(\frac{1}{\sqrt{Np}} \sum_{k=0}^{p-1} e^{2\pi i (x_0 + kr) y/N} \right) |y\rangle. \tag{8.51}$$

This is a superposition of basis states with a probability of occurrence of a particular y given by the mod-squared of the term in the brackets:

$$\begin{aligned} \mathcal{P}(y) &= \frac{1}{Np} \left| \sum_{k=0}^{p-1} e^{2\pi i (x_0 + kr)/N} \right|^2 \\ &= \frac{1}{Np} \left| \sum_{k=0}^{p-1} e^{2\pi i k r y/N} \right|^2 \end{aligned} \tag{8.52}$$

So y has an r-dependent probability of occurrence. The crux of this algorithm is that the most probable value of y gives us enough information about r for us to compute it. In fact, the claim is that the values of y that are measured are close to an integer multiple of N/r.

Let's first see this in the special case when there are exactly integer number of periods in the interval $[0, N-1]$, i.e., when

$$p = \frac{N}{r}.$$

We will compare the probability of y for mp when m is some integer, and when not:

$$\mathcal{P}(y) = \frac{1}{r}\left|\frac{1}{p}\sum_{k=0}^{p-1} e^{2\pi iky/p}\right|^2. \tag{8.53}$$

$$\mathcal{P}(y = mp) = \frac{1}{r}, \tag{8.54}$$

$$\mathcal{P}(y \neq mp) = \frac{1}{rp^2}\left|\sum_{k=0}^{p-1} e^{ik\theta}\right|^2, \text{ where } \theta = 2\pi\frac{y}{p}, \tag{8.55}$$

$$= \frac{1}{rp^2}\frac{\sin^2(p\theta/2)}{\sin^2(\theta/2)}$$

$$= 0 \text{ (since } p\theta \text{ is an integer multiple of } 2\pi). \tag{8.56}$$

So the *only* values of y obtained in this case are integer multiples of N/r.

For a general function, it is highly unlikely that there are exactly integer numbers of periods in the interval $[0, N-1]$. Yet, the most probable values of y turn out to be close to integer multiples of N/r! To see this, let us start by writing

$$y = m\frac{N}{r} + \delta_m, \tag{8.57}$$

where m is an integer and $|\delta_m| \leq \frac{1}{2}$. Let's substitute this in Equation 8.52:

$$\mathcal{P}(y) = \frac{1}{Np}\left|\sum_{k=0}^{p-1} e^{2\pi ikr(mN/r+\delta_m)/N}\right|^2$$

$$= \frac{1}{Np}\left|\sum_{k=0}^{p-1} e^{2\pi ikr\delta_m/N}\right|^2$$

$$= \frac{1}{Np}\frac{\sin^2(p\theta_m)}{\sin^2\theta_m}, \text{ where } \theta_m = \frac{\pi r}{N}\delta_m. \tag{8.58}$$

Now since p is nearly N/r, the numerator is nearly $\sin^2(\pi\delta_m)$. Also, $r\delta_m/N$ is very small, so the denominator is nearly $\theta_m \sim \pi\delta_m r/N$.

Therefore,

$$\mathcal{P}(y) \sim \frac{1}{Np}\frac{\sin^2(\pi\delta_m)}{(\pi\delta_m r/N)^2} = \frac{1}{r}\frac{\sin^2(\pi\delta_m)}{(\pi\delta_m)^2}. \tag{8.59}$$

Since $\delta_m < 1/2$, and $\dfrac{\sin\theta}{\theta} \geq \dfrac{2}{\pi}$ for $0 \leq \theta \leq \pi/2$, (see Figure 8.12), we have

$$\mathcal{P}(y \sim m/r) \geq \frac{4}{\pi^2 r}. \tag{8.60}$$

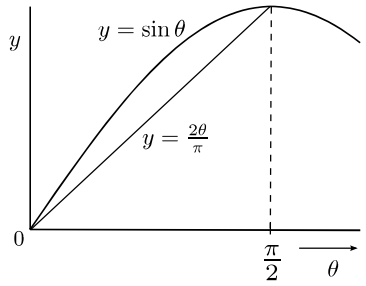


FIGURE 8.12: Graph comparing $\sin\theta$ and $2\theta/\pi$.

There are r possible such y's, so the probability of any such y is greater than $4/\pi^2 \sim 40\%$. This result is to be interpreted as saying that when we rerun the algorithm many times, with high probability we measure y's that are integer multiples of N/r. Now from such numbers we can use classical algorithms to deduce r, most famously the Euclid algorithm for continued fractions. The period-finding algorithm thus succeeds to a high probability.

Such analyses of the probability of obtaining good results are a common feature of most known quantum algorithms.

Box 8.3: Finding r Given N/r: Continued Fractions

The output y of a run of the period-finding algorithm is close to an integer multiple of N/r. Consider the number $x = y/N \sim m/r$. We now look at the continued fraction expansion of x:

$$x = c_0 + \frac{1}{x_1} = c_0 + \frac{1}{c_1 + \frac{1}{x_2}} = c_0 + \frac{1}{c_1 + \frac{1}{c_2 + \frac{1}{x_3}}} = \cdots \tag{8.61}$$

$$= c_0 + \cfrac{1}{c_1 + \cfrac{1}{c_2 + \cfrac{1}{c_3 + \cdots}}} \tag{8.62}$$

At each stage of the expansion (Equations 8.61), c_i is the integer part of the denominator x_i from the previous stage, and each x_i, known as the i^{th} *partial sum*, is a fraction $\in [0, 1]$. To find the fractional expression for $\frac{1}{x_i}$, Euclid's GCD algorithm can be used. Equation 8.62 is the continued fraction expansion of x. If x is a rational number then the continued fraction expansion terminates after a finite number of steps. For n-bit m and r, it turns out that the continued fraction can be computed in $\mathcal{O}(n^3)$ steps.

Now there is a theorem (proved in [50], Appendix 4) stating that m/r is

one of the partial sums x_i of the continued fraction of x. $r < N$, and the best guess for r is the partial sum having the largest denominator less than N. This is tested out and if it is not the period then the we try again with a different x.

8.5.1.1 Shor's factorization algorithm

The above algorithm for period finding, due in some form to Peter Shor, is really the heart of the factorization algorithm. For the more curious, the relationship between factoring and period-finding is through a series of mathematical results that we will outline here. (This section is purely for the purpose of completeness, and the results of pure mathematics used will not be derived or explained.)

For a good understanding of what follows, one must be familiar with *modular algebra*, that is algebra restricted to the range $[0, N-1]$ by considering all results of algebraic operations as periodic with period N. Then "mod N" essentially means "the remainder after dividing the result by N". For example, addition mod 4 will mean $2+2=0$ and $2+3=1$.

- If a is a random integer $< N$ such that a and N are coprime, then it is possible to find an integer $r \in [1, N]$ such that

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

  r is called the *order* of a in $\bmod N$.

- For a with order r mod N, the function

  $$f(x) = a^x \bmod N,$$

  is periodic with period r. To see how:

  $$\begin{aligned} f(x+r) &= a^{x+r} \bmod N = (a^x \bmod N)(a^r \bmod N) \\ &= a^x \bmod N \times 1 = f(x). \end{aligned}$$

  Therefore, finding the period of a function $f(x)$ is the same as finding the order of some integer coprime with N.

- Now if N is a large integer, choose a random integer a coprime with N and find its order r using the period-finding algorithm. Now if r is even then construct $b = a^{r/2}$.

  $$\begin{aligned} b^2 &= 1 \bmod N \\ \implies b^2 - 1 &= 0 \bmod N \end{aligned}$$

  So $b \pm 1$ must have factors common with N. If we find the GCDs of $b \pm 1$ and N we have the prime factors of N!

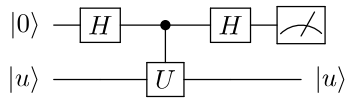
8.5.2 Phase estimation

One version of Shor's algorithm is based on phase estimation. This application of the quantum Fourier transform is used to estimate the eigenvalue of a unitary operator, which is a phase:

$$\hat{U}|u\rangle = e^{i\theta}|u\rangle; \quad \theta = 2\pi\phi \tag{8.63}$$

where ϕ is a fraction.

As a preliminary to this algorithm, let's look at a toy version. Suppose you are given U and an eigenstate $|u\rangle$. We have seen that the circuit of Figure 7.14 simulates a measurement of U.



Here,

$$\frac{1}{\sqrt{2}}\big(|0\rangle + |1\rangle\big) \otimes |u\rangle \rightarrow \frac{1}{\sqrt{2}}\big(|0\rangle + e^{2\pi i\phi}|1\rangle\big) \otimes |u\rangle. \tag{8.64}$$

If ϕ were a single bit, then you can see that the output is 0 if $\phi = 0.0$ and 1 if $\phi = 0.1$. This circuit thus gives us the value in one run. But in general ϕ will be several bits long. A measurement of the upper register in the H basis will yield a 0 or 1 with probabilities $\cos^2 \pi\phi$ and $\sin^2 \pi\phi$. A statistically large number of measurements will allow us to recover ϕ from the counts. But this is an inefficient method.

Note that the H transform on the upper register is the one-bit Fourier transform. In order to estimate ϕ to more bits of accuracy, we must have a qubit for each significant figure of ϕ and then perform an inverse Fourier transform, as shown in the circuit of Figure 8.13.

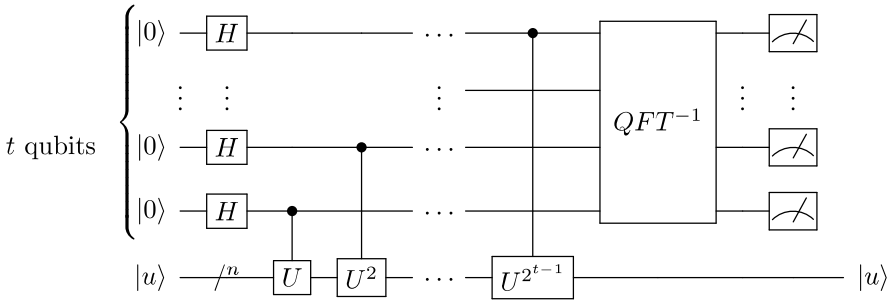


FIGURE 8.13: Circuit for phase estimation.

Imagine ϕ upto t bits as

$$\phi = 0.\phi_1\phi_2\cdots\phi_t = \frac{\phi'}{2^t}, \quad \phi' = \phi_t\phi_{t-1}\cdots\phi_1. \tag{8.65}$$

Then we start with t working qubits in the input register, and use them to control gates of the form U^{2^k}. After the control gates, the output on the k^{th} line is

$$\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \longrightarrow \frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i 2^k \phi}|1\rangle\right) \tag{8.66}$$

$$= \frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i (0.\phi_1\phi_2\cdots\phi_k)}|1\rangle\right). \tag{8.67}$$

You can see that just before the QFT gate, the state of the upper register is

$$\frac{1}{\sqrt{2^t}}\left(|0\rangle + e^{2\pi i\phi}|1\rangle\right) \otimes \left(|0\rangle + e^{4\pi i\phi}|1\rangle\right) \otimes \ldots \otimes \left(|0\rangle + e^{2\pi i.2^t\phi}|1\rangle\right) \tag{8.68}$$

$$= \frac{1}{\sqrt{2^t}} \sum_{k=0}^{2^t-1} e^{2\pi i\phi k}|k\rangle \tag{8.69}$$

This is just the QFT mod 2^t of ϕ' and an inverse Fourier transform will give you ϕ' exact to t significant figures.

8.6 Grover's Search Algorithm

Another famous algorithm that made a splash in the world of quantum computing was invented by L. K. Grover in 1997 [40]. This algorithm is in a different class from the ones we have studied so far, which may all be said to be QFT-based. Grover's algorithm introduced a new technique: amplitude amplification. Even though it did not demonstrate an exponential speed-up over the classical case, it was still dramatic enough to get noticed.

The problem Grover attacked was that of search for an element in an unstructured database. The problem is like doing a reverse search in a phone directory: you have a number and need to know the person it belongs to. Thus there is no regular short-cut to the search, you have to go through each entry in the book and check if it matches the number you have.

The problem can be phrased in the language of oracles, if we assume that the criterion for the search is built into a function evaluator: a function that tells you whether the input number matches the search criterion or not. So we imagine that the numbers x are indices to the entries in the database, and one index, let's say k, belongs to the entry being searched for. Then

$$f_k(x) = \begin{cases} 1 & \text{if } x = k \\ 0 & \text{otherwise.} \end{cases} \tag{8.70}$$

Here, if x is an n-bit number, then the size of the database is $2^n = N$. As

this becomes really large, the problem becomes harder. In fact, classically this problem has a complexity $\mathcal{O}(N)$. Grover's algorithm turns out to be $\mathcal{O}(\sqrt{N})$.

As in most quantum algorithms, the first step exploits quantum parallelism, and inputs the uniform superposition of all x's to the oracle U_{f_k}, the unitary implementation of $f_k(x)$:

$$|\psi\rangle = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{N-1} |x\rangle. \tag{8.71}$$

It also uses the phase kickback trick to give f-dependent phases to the states in this superposition:

$$|\psi\rangle \otimes |-\rangle \xrightarrow{U_{f_k}} \frac{1}{\sqrt{2^{n+1}}} \sum_{x=0}^{N-1} (-1)^{f_k(x)} \; |x\rangle \otimes |-\rangle. \tag{8.72}$$

We need to look more closely at the form of the output state. Remember that $f_k(x)$ is zero unless $x = k$. Thus each $|x\rangle$ in the above superposition has the same phase $(+1)$ as before except the state $|k\rangle$ which has a phase -1. Thus though we do not know what k is, this step is equivalent to *tagging* that particular state:

$$\sum_{x=0}^{N-1} (-1)^{f_k(x)} \; |x\rangle = |0\rangle + |1\rangle + \cdots - |k\rangle + \cdots + |N-1\rangle. \tag{8.73}$$

Algebraically, this step is equivalent to the action of the following *oracle operator* on the input register alone:

$$\hat{O} = \mathbb{1} - 2|k\rangle\langle k|, \tag{8.74}$$

since $|k\rangle\langle k|$ projects the state $|k\rangle$ out of the superposition. It helps to visualize this step, as well as the rest of the algorithm, by looking at what happens to the input state $|\psi\rangle$ in the Hilbert space $\mathcal{H}^{\otimes n}$. This space is spanned by the n unit vectors $\{|x\rangle\}$. Concentrate on the 2-d hyperplane spanned by the solution ket $|k\rangle$ and the vector $|\alpha\rangle$, a linear combination of all the other basis states, representing the hyperplane perpendicular to $|k\rangle$. You can visualize the input state $|\psi\rangle$ in this plane, as having a (small) component $\frac{1}{\sqrt{N}}$ along $|k\rangle$. The oracle operator $\hat{O}$ of Equation 8.74) reverses the sign of this component, performing a reflection in the hyperplane $|\alpha\rangle$ as shown in Figure 8.14.

Check out what this figure shows us. The initial uniform superposition is equally "far" from all the basis kets, including the target $|k\rangle$. In fact, it makes an angle

$$\theta = \sin^{-1} \frac{1}{\sqrt{N}} \tag{8.75}$$

with $|\alpha\rangle$ in this plane. The idea behind Grover's algorithm is to increase this

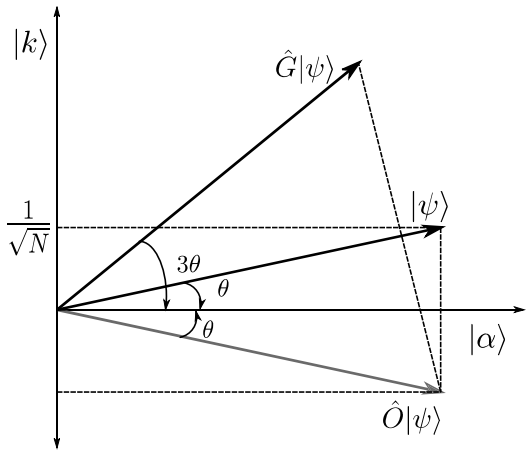


FIGURE 8.14: Geometric Visualization of the action of the Grover Iterate

angle to $\frac{\pi}{2}$ or as close to it as possible, by manipulating the phases of the state of the quantum computer.

Grover took a clue from the action of the operator $\hat{\mathcal{O}}$, and came up with another one $\hat{S}$, which implements a reflection in the plane of $|\psi\rangle$, which brings the input state closer to $|k\rangle$. By iterating this process a sufficient number of times, the input state $|\psi\rangle$ is rotated to $|k\rangle$. To see how this happens, let's construct $\hat{S}$:

$$\begin{aligned} \hat{S} &= \mathbb{1} - 2\big(\mathbb{1} - 2|\psi\rangle\langle\psi|\big) \\ &= 2|\psi\rangle\langle\psi| - \mathbb{1}. \end{aligned} \tag{8.76}$$

The action of the Grover iterate $\hat{G} = \hat{S}\hat{O}$ is to rotate the input state by an angle 3θ towards $|k\rangle$. This can be seen in the basis of vectors $|k\rangle$ and $|\alpha\rangle$:

$$\hat{G}|\psi\rangle = \cos 3\theta|\alpha\rangle + \sin 3\theta|k\rangle. \tag{8.77}$$

If this is repeated p times,

$$\hat{G}^p|\psi\rangle = \cos[(2p+1)\theta]|\alpha\rangle + \sin[(2p+1)\theta]|k\rangle. \tag{8.78}$$

Thus the iteration can stop when

$$\begin{aligned} (2p+1)\theta &\sim \frac{\pi}{2} \\ p &\sim \frac{1}{2\theta}\left(\frac{\pi}{2} - \theta\right). \end{aligned} \tag{8.79}$$

For large N, we have $\sin\theta = \frac{1}{\sqrt{N}} \simeq \theta$, so that the number of iterations required is given by

$$p = \frac{\pi}{4}\sqrt{N} - \frac{1}{2} = \mathcal{O}(\sqrt{N}). \tag{8.80}$$

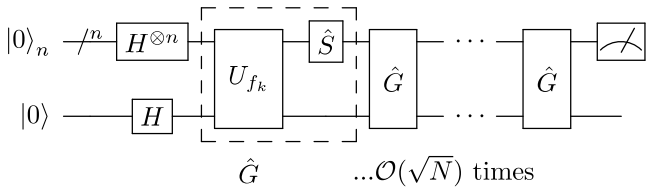


FIGURE 8.15: Circuit implementing Grover's algorithm

The circuit representation for this algorithm is given in Figure 8.15.

The process of rotating the superposition $|\psi\rangle$ towards the solution essentially works because of increasing the amplitude for $|k\rangle$, so this method goes under the name of "amplitude amplification".

Example 8.6.1. Let's look at the case $N = 4$, for 2-bit indices. The initial angle is given by $\sin\theta = \frac{1}{2} \implies \theta = \frac{\pi}{6}$. After a single iteration, the angle becomes $\frac{\pi}{2}$: thus a single run of the algorithm gives the answer.

Example 8.6.2. The deity who constructs the oracle in Grover's algorithm must give us a circuit implementing U_f for the checking the criterion. Let's say we have a 5-bit database and the 19^{th} entry is the search item. In binary, the index for the solution is $k = 18 = 10010$. The oracle output must be 1 for this input and 0 otherwise. It's easy to see that the circuit of Figure 8.16 will do the trick:

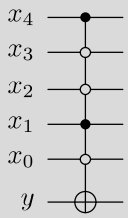


FIGURE 8.16: Construction of oracle for $k = 18$.

Example 8.6.3. We'll see how to construct the circuit for $\hat{S}$ (Equation 8.76).

$$\begin{aligned}\hat{S} &= -(\mathbb{1} - 2|\psi\rangle\langle\psi|) \\ &= -H^{\otimes n}\left(\mathbb{1} - 2|0\rangle\langle 0|\right)H^{\otimes n} \\ &= H^{\otimes n}\hat{P}H^{\otimes n}\end{aligned}$$

where the operator $\hat{P}$ leaves all basis states unchanged except $|0\rangle$, whose

sign is flipped. (We can also ignore the overall negative sign.) This can be implemented by an $(n-1)$-fold 0-controlled Z gate. Since $Z = HXH$, we have the circuit of Figure 8.17 for $\hat{S}$.

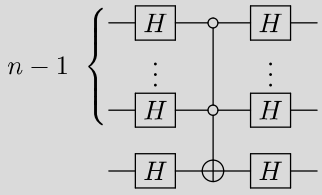


FIGURE 8.17: Construction of operator $\hat{S}$.

8.6.1 Extension to multiple solutions

A simple extension to this algorithm works when the search criterion has multiple solutions. The oracle then gives an answer "yes" whenever any one of the M possible solutions is input. The Hilbert space then has a "solution subspace" $\mathcal{M}$, spanned by M solution states. Let us denote by $|\beta\rangle$ the uniform superposition of all these vectors, and by $|\alpha\rangle$, its orthogonal complement.

$$|\beta\rangle = \frac{1}{\sqrt{M}} \sum_{x \in \mathcal{M}} |x\rangle, \tag{8.81}$$

$$|\alpha\rangle = \frac{1}{\sqrt{N-M}} \sum_{x \notin \mathcal{M}} |x\rangle. \tag{8.82}$$

In this situation, the input state is

$$|\psi\rangle = \sqrt{\frac{M}{N}}|\beta\rangle + \sqrt{\frac{N-M}{N}}|\alpha\rangle. \tag{8.83}$$

The angle θ is now given by

$$\sin\theta = \langle\psi|\beta\rangle = \sqrt{\frac{M}{N}}. \tag{8.84}$$

The operator $\hat{O}$ tags all of the M solutions with a '$-$' sign, and the Grover iterate is defined the same way as before. After p iterations we get the state

$$\hat{G}^p|\psi\rangle = \cos[(2p+1)\theta]|\alpha\rangle + \sin[(2p+1)\theta]|\beta\rangle, \tag{8.85}$$

and the number of iterations required is nearly

$$p \approx \frac{\pi}{4}\sqrt{\frac{N}{M}}. \tag{8.86}$$

You should check for yourself that this works out.

8.6.2 Quantum counting

A fallout of the multiple search algorithm is the counting algorithm. Before we know how many times to iterate, we need to know how many solutions M there are to the search criterion. Can we deduce a quantum algorithm for finding M given U_{f_k}? The solution found by Brassard et al. [15], is a combination of Grover's search and Shor's phase estimation algorithms. The key point here is to note that the number of solutions is related to the eigenvalues of the operator $\hat{G}$, which can also be expressed in the $|\alpha\rangle$-$|\beta\rangle$ basis as the 2-d matrix

$$\hat{G} = \begin{pmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{pmatrix} \text{ where } \sin\varphi = 2\frac{\sqrt{M(N-M)}}{N}. \tag{8.87}$$

The eigenvalues of this matrix are $e^{\pm i\varphi}$. We can therefore use the phase estimation algorithm to deduce φ. We need to feed the algorithm with an eigenstate of $\hat{G}$. Now you can see for yourself that $|\psi\rangle$ is a linear superposition of the two eigenstates of $\hat{G}$, so the circuit of Figure 8.18 will work to t bits of accuracy.

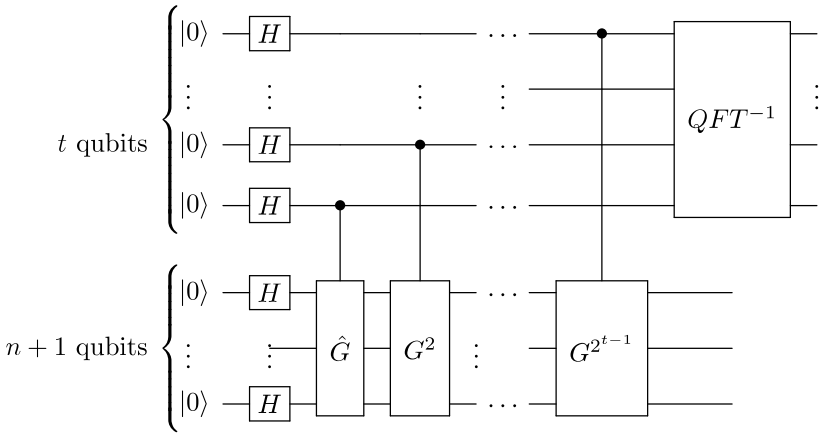


FIGURE 8.18: Circuit for quantum counting.

For Further Reading

The subject of quantum algorithms has rapidly evolved from its beginnings. Good references, apart from the original papers that may be found in the bibliography, are the excellent text book by Kaye, Laflamme, and Mosca [43] and the introductory text by Rieffel and Polak [58].

Problems

8.1. Some texts implement the quantum function evaluator as a "controlled-$\tilde{U}_f$" gate (Figure 8.19), where $\tilde{U}_f$ acts only on the lower register, and is defined by $\tilde{U}_f|y\rangle = |y \oplus f(x)\rangle$:

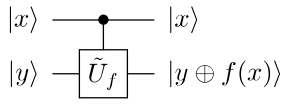


FIGURE 8.19: The quantum function evaluator as a controlled $\tilde{U}_f$ gate.

How is the action of this implementation different from the f-controlled NOT gate of Figure 7.15? Check by using standard basis states as well as superpositions as inputs.

8.2. Show that the phase kickback trick works because the input state in the bottom register is an eigenstate of the $\tilde{U}_f$ operator for the Deutsch algorithm.

8.3. Deutsch's original version of his algorithm used $|0\rangle$ as the input to the bottom register instead of $|0\rangle - |1\rangle$. Show that in this case you obtain the correct answer with probability $3/4$. Also show that the algorithm has probability $1/2$ of succeeding.

8.4. Prove the shift-invariance property of the Fourier transform, i.e., show that

$$\hat{\mathcal{F}}|x + k\rangle = e^{i\theta}\hat{\mathcal{F}}|x\rangle \tag{8.88}$$

for some θ. Find θ in terms of k.

8.5. For the operator R_d of Equation 8.45, give a construction for the controlled R_d gate using CNOT and single-qubit gates.

8.6. Find the eigenvalues and eigenvectors of the matrix R_d. What can you say about the commutators (i) $[R_d, X]$ (ii) $[R_d, Y]$ (iii) $[R_d, Z]$ (iv) $[R_d, R'_d]$?

8.7. Work out a circuit that calculates the inverse quantum Fourier transform.

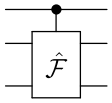
8.8. Consider a periodic function $f(x + r) = f(x)$ for $0 \leq x < N$ where N is

an integer multiple of r. Suppose you are given a unitary operator U_y that performs the transformation $U_y|f(x)\rangle = |f(x+y)\rangle$. Show that the state

$$|\tilde{f}(k)\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} e^{-2\pi i k x/N} |f(x)\rangle \tag{8.89}$$

is an eigenvector of U_y. Calculate the corresponding eigenvalue.

8.9. Compute the output of the controlled-QFT gate shown in the figure if the input is $H^{\otimes 3}|x\rangle$.



8.10. On examining the period finding algorithm, we can find a relationship with the phase-estimation algorithm. On applying the oracle, we get

$$\frac{1}{\sqrt{N}} \sum |x\rangle|0\rangle \xrightarrow{U_f} \frac{1}{\sqrt{N}} \sum |f(x)\rangle.$$

Express $|f(x)\rangle$ in terms of its Fourier transform, $|\tilde{f}(k)\rangle$. Invert this expression and show that $|\tilde{f}(k)\rangle$ are of the same form as Equation 8.89 of Problem 8.8. Now show that the period finding algorithm is the phase estimation for the operator U_y defined there.

8.11. Apply the quantum phase estimation algorithm to the following cases and obtain the results:

(a) $U = X, \quad |u\rangle = |-\rangle, \quad t = 2,$

(b) $U = R_d, \quad |u\rangle = |1\rangle, \quad t = d+1.$