ELEN 162
A. I. Zecevic

Lecture Notes for Week 4

In contrast to traditional computers (which operate on *bits* of information), quantum computers work with *qubits*. Qubits can be identified with quantum particles whose state has the general form $\psi = \alpha\psi_0 + \beta\psi_1$, where ψ_0 and ψ_1 correspond to observable values of some physical quantity. The specific interpretation of states ψ_0 and ψ_1 depends on the physical realization of the system. It can refer, for example, to the spin of an atom (in which case 0 and 1 correspond to "up" and "down" orientations along an axis), to its energy state (where 0 and 1 represent the "ground" and the "excited" state, respectively), or to any number of other physical properties with binary characteristics.

Although such a definition directly associates qubits with physical properties of quantum particles, for our purposes it will be more convenient to think of a qubit as an abstract mathematical entity. The advantage of such an approach is its inherent generality, which allows us to explain the principles of quantum computing in a way that is independent of the specific implementation.

We will begin our discussion by examining how qubits differ from classical bits. One of the most important distinctions stems from the fact that the state of a qubit is *not* limited to ψ_0 and ψ_1 (which correspond to the "classical" 0 and 1). While it is true that we will record one of these two states when we perform a measurement, a qubit can also be in a *state of superposition*, in which both α and β are nonzero. There is an unlimited number of such "intermediate" states, since the only constraint that these coefficients must satisfy is $|\alpha|^2 + |\beta|^2 = 1$.

It is important to recognize in this context that the information contained in the state of superposition is "hidden", since any measurement performed on such a system will necessarily produce either ψ_0 or ψ_1 (with probabilities $|\alpha|^2$ and $|\beta|^2$, respectively). These two states tell us nothing about what the function looked like before the observation was made - it could have been in *any* state $\psi = \alpha\psi_0 + \beta\psi_1$. What is interesting, however, is that quantum computing can exploit this "hidden" information in a very elegant way.

To explain how that can be done, we first need to describe the basic building blocks of quantum computers (which are known as *quantum gates*). As the name suggests, quantum gates have certain similarities with standard logic gates that are used in the design of digital computers. There are, however, some fundamental differences as well, one of which has to do with the fact that quantum gates operate on the coefficients that describe the state of superposition (rather than the measurable states themselves).

To see why this is an advantage, we should recall that a standard logic gate with n inputs can perform only *one* operation at a time (an n-input AND gate, for example, transforms any input combination $X_1, X_2, \ldots, X_n$ into $Z = X_1 \cdot X_2 \cdot \ldots \cdot X_n$). An n-input quantum gate, on the other hand, operates on all 2^n coefficients $\{a_0, a_1, \ldots, a_{2^n-1}\}$ in expression

$$\Psi = \sum_{k=0}^{2^n-1} a_k \Psi_k \tag{1}$$

simultaneously, and transforms them into a different set of coefficients $\{\rho_0, \rho_1, \ldots, \rho_{2^n-1}\}$.

Single Qubit Gates

In the following, we will consider three types of single qubit gates, each of which plays an important role in quantum computing.

The Quantum NOT Gate

The quantum NOT gate corresponds to operator $\hat{X}$, which we already encountered. This operator was defined using relations

$$\hat{X}\psi_0 = \psi_1 \tag{2}$$

and

$$\hat{X}\psi_1 = \psi_0 \tag{3}$$

and we established that its matrix representation in basis $\{\psi_0, \psi_1\}$ is

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \tag{4}$$

Matrix X is useful because it allows us to easily compute function $\varphi = \hat{X}\psi$ for any given choice of ψ. We can do so since applying operator $\hat{X}$ to function $\psi = \alpha_0\psi_0 + \alpha_1\psi_1$ produces

$$\varphi = \hat{X}\psi = \rho_0\psi_0 + \rho_1\psi_1 \tag{5}$$

and we know that coefficients ρ_0 and ρ_1 are related to α_0 and α_1 as

$$\begin{bmatrix} \rho_0 \\ \rho_1 \end{bmatrix} = X \cdot \begin{bmatrix} \alpha_0 \\ \alpha_1 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} \alpha_0 \\ \alpha_1 \end{bmatrix} \tag{6}$$

It is not difficult to show that this operator satisfies $\hat{X}^2 = \hat{I}$. Indeed, if we apply $\hat{X}$ to function ψ *twice*, we obtain

$$\begin{aligned} &\xi = \hat{X}^2\psi = \hat{X}(\hat{X}\psi) = \hat{X}[\alpha_0(\hat{X}\psi_0) + \alpha_1(\hat{X}\psi_1)] = \hat{X}(\alpha_0\psi_1 + \alpha_1\psi_0) = \\ &= \alpha_0(\hat{X}\psi_1) + \alpha_1(\hat{X}\psi_0) = \alpha_0\psi_0 + \alpha_1\psi_1 = \psi \end{aligned} \tag{7}$$

Since repeating this transformation returns the qubit to its original state, we say that the quantum NOT gate is *reversible*.

It is important to recognize that this gate is more general than its classical counterpart, since it *simultaneously* manipulates both α_0 and α_1 (rather than a single 0 or 1), and turns them into ρ_0 and ρ_1. The two operations match only if ($\alpha_0 = 1$, $\alpha_1 = 0$) or ($\alpha_0 = 0$, $\alpha_1 = 1$), in which case the quantum transformations

$$\hat{X}\psi_0 = \psi_1 \tag{8}$$

and

$$\hat{X}\psi_1 = \psi_0 \tag{9}$$

reduce to simple "bit flips" (of the sort that a standard NOT gate performs). This is a significant difference, since it implies that quantum gates can produce an unlimited number of internal states *beyond* the ones that are actually measurable. Although we cannot observe them, these states represent an essential part of the computation.

The Z - Gate

The operator that corresponds to the Z-gate is defined by relations

$$\hat{Z}\psi_0 = \psi_0 \tag{10}$$

and

$$\hat{Z}\psi_1 = -\psi_1 \tag{11}$$

To determine its matrix representation, let us assume that this operator transforms function $\psi = \alpha_0\psi_0 + \alpha_1\psi_1$ into

$$\varphi = \hat{Z}\psi = \rho_0\psi_0 + \rho_1\psi_1 \tag{12}$$

If we now rewrite $\hat{Z}\psi_0$ and $\hat{Z}\psi_1$ as

$$\hat{Z}\psi_0 = a_{11}\psi_0 + a_{12}\psi_1 \tag{13}$$

and

$$\hat{Z}\psi_1 = a_{21}\psi_0 + a_{22}\psi_1 \tag{14}$$

expressions (10) and (11) imply that

$$\begin{aligned} a_{11} &= 1 \\ a_{12} &= 0 \\ a_{21} &= 0 \\ a_{22} &= -1 \end{aligned} \tag{15}$$

We can therefore conclude that the matrix representation of operator $\hat{Z}$ has the form

$$Z = \begin{bmatrix} a_{11} & a_{21} \\ a_{12} & a_{22} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \tag{16}$$

It is easily verified that this operator satisfies $\hat{Z}^2 = \hat{I}$ (just like operator $\hat{X}$), which means that the quantum Z-gate is reversible.

The Hadamard Gate

The third type of single qubit gate that will be of interest to us is the so-called Hadamard gate, which is associated with the Hadamard operator. We previously established that this operator is defined by relations

$$\hat{H}\psi_0 = \frac{1}{\sqrt{2}}\psi_0 + \frac{1}{\sqrt{2}}\psi_1 \equiv \psi_+ \tag{17}$$

and

$$\hat{H}\psi_1 = \frac{1}{\sqrt{2}}\psi_0 - \frac{1}{\sqrt{2}}\psi_1 \equiv \psi_- \tag{18}$$

and that its matrix representation has the form

$$H = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{19}$$

As in the previous two cases, it turns out that the corresponding quantum gate is reversible, since $\hat{H}$ satisfies $\hat{H}^2 = \hat{I}$.

Some Additional Properties of the Hadamard Operator

One of the reasons why the Hadamard gate is important in quantum computing has to do with the fact that it produces functions $\hat{H}\psi_0 = \psi_+$ and $\hat{H}\psi_1 = \psi_-$, respectively. This turns out to be a very useful property, paticularly in dealing with quantum algorithms that involve multiple qubits. To get a sense for why this is so, let us consider what happens when we apply the Hadamard operator to a two particle system that is in state $\psi_0 \otimes \psi_0$. If we do that, we obtain

$$\begin{aligned}(\hat{H} \otimes \hat{H})(\psi_0 \otimes \psi_0) = (\hat{H}\psi_0) \otimes (\hat{H}\psi_0) = \frac{1}{\sqrt{2}}(\psi_0 + \psi_1) \otimes \frac{1}{\sqrt{2}}(\psi_0 + \psi_1) = \\ = \frac{1}{2}\left[(\psi_0 + \psi_1) \otimes (\psi_0 + \psi_1)\right] = \frac{1}{2}\left[\Psi_{00} + \Psi_{01} + \Psi_{10} + \Psi_{11}\right]\end{aligned} \tag{20}$$

(using properties of tensor products). Note that all four outcomes have the *same* probability in this case, which is something that we will frequently exploit in our discussion of quantum algorithms.

In applications of this sort, operator $\hat{H} \otimes \hat{H}$ is usually expressed as $\hat{H}^{\otimes 2}$ (in order to simplify the notation). This allows us to represent $(\hat{H} \otimes \hat{H})(\psi_0 \otimes \psi_0)$ as $\hat{H}^{\otimes 2}(\Psi_{00})$ (which is much more compact), and do something similar in cases when the number of particles is larger. To illustrate how this works in practice, suppose that we have 3 qubits in state ψ_0, and that we decided to apply a Hadamard operator to each of them. Using expression (20), the resulting function $(\hat{H} \otimes \hat{H} \otimes \hat{H})(\psi_0 \otimes \psi_0 \otimes \psi_0)$ can then be represented as

$$\begin{aligned}(\hat{H} \otimes \hat{H} \otimes \hat{H})(\psi_0 \otimes \psi_0 \otimes \psi_0) = (\hat{H}\psi_0) \otimes (\hat{H}\psi_0) \otimes (\hat{H}\psi_0) = \\ = \psi_+ \otimes (\psi_+ \otimes \psi_+) = \frac{1}{\sqrt{2}}(\psi_0 + \psi_1) \otimes \frac{1}{2}(\Psi_{00} + \Psi_{01} + \Psi_{10} + \Psi_{11})\end{aligned} \tag{21}$$

Setting

$$\Psi_{ijk} = \psi_i \otimes \Psi_{jk} = \psi_i \otimes (\psi_j \otimes \psi_k) \tag{22}$$

and applying the notation that we just introduced, expression (21) can be rewritten as

$$\hat{H}^{\otimes 3}(\Psi_{000}) = \frac{1}{\sqrt{2^3}}(\Psi_{000} + \Psi_{001} + \ \ldots\ + \Psi_{110} + \Psi_{111}) \tag{23}$$

It is not difficult to see that we now have 8 possible outcomes, each of which is equally likely.

This approach can obviously be generalized to n qubits that are in state ψ_0. In that case, we obtain a composite wave function with 2^n components, which be represented as

$$\hat{H}^{\otimes n}(\Psi_0) = \frac{1}{\sqrt{2^n}} \sum_{x\,=\,0}^{2^n-1} \Psi_x \tag{24}$$

Such generalizations are important because a number of quantum algorithms use $\hat{H}^{\otimes n}(\Psi_0)$ as their starting point. Functions Ψ_x that appear in (24) should be interpreted as $\Psi_0 = \Psi_{00\ldots0}$,

$\Psi_1 = \Psi_{00\ldots1}$, $\ldots$, $\Psi_{2^n-1} = \Psi_{11\ldots1}$. Note that this expression has the same general form as expressions (20) and (23), except for the fact that each state Ψ_x now has probability

$$P(x) = \left(\frac{1}{\sqrt{2^n}}\right)^2 = \frac{1}{2^n} \tag{25}$$

When analyzing the properties of operator $\hat{H}$, we should also mention that it has an alternative representation which is often used in quantum computing. To see what this representation looks like, it suffices to observe that expressions

$$\hat{H}\psi_0 = \frac{1}{\sqrt{2}}\psi_0 + \frac{1}{\sqrt{2}}\psi_1 \tag{26}$$

and

$$\hat{H}\psi_1 = \frac{1}{\sqrt{2}}\psi_0 - \frac{1}{\sqrt{2}}\psi_1 \tag{27}$$

can be rewritten in the form

$$\hat{H}\psi_x = \frac{1}{\sqrt{2}}[\psi_0 + (-1)^x\psi_1] \tag{28}$$

where x can take values 0 or 1 (it is hopefully obvious that we can obtain (26) by setting $x = 0$, and that (27) corresponds to $x = 1$). This result will prove to be very useful in developing a quantum algorithm for eigenvalue estimation.

Multiple Qubit Gates

Perhaps the most important multiple qubit gate is the so-called C-NOT gate, whose schematic diagram is shown in Fig. 1. This type of gate is said to be *universal* because it can be used to construct *any* other multiple qubit gate.

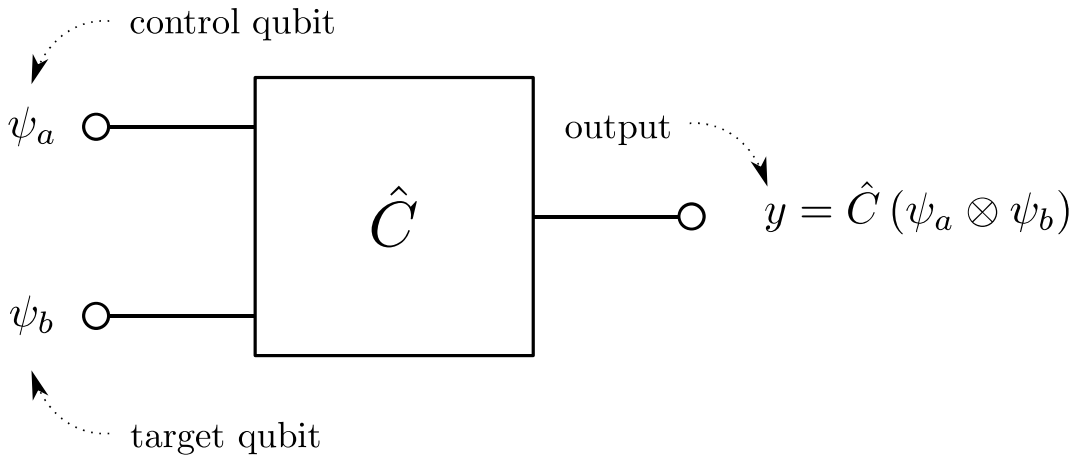

Figure 1: Schematic representation of a C-NOT gate.

The operator $\hat{C}$ that is associated with this type of gate acts on the four basis states $\{\Psi_{00}, \Psi_{01}, \Psi_{10}, \Psi_{11}\}$ in the following way

$$\begin{aligned} \hat{C}(\Psi_{00}) &= \Psi_{00} \\ \hat{C}(\Psi_{01}) &= \Psi_{01} \\ \hat{C}(\Psi_{10}) &= \Psi_{11} \\ \hat{C}(\Psi_{11}) &= \Psi_{10} \end{aligned} \tag{29}$$

Note that this transformation changes function Ψ_{ij} only if the *first bit in the index is* 1 (in which case, the second bit "flips").

To get a sense for what this means in practice, let us assume that the control qubit is in state $\psi_a = \alpha_0\psi_0 + \alpha_1\psi_1$ and that the target qubit is in state $\psi_b = \beta_0\psi_0 + \beta_1\psi_1$. When these two qubits interact, the tensor product $\psi_a \otimes \psi_b$ assumes the familiar form

$$\psi_a \otimes \psi_b = a_0\Psi_{00} + a_1\Psi_{01} + a_2\Psi_{10} + a_3\Psi_{11} \tag{30}$$

When operator $\hat{C}$ is applied to this expression, we obtain a new function

$$\begin{aligned} &\Phi = \hat{C}(\psi_a \otimes \psi_b) = a_0\hat{C}(\Psi_{00}) + a_1\hat{C}(\Psi_{01}) + a_2\hat{C}(\Psi_{10}) + a_3\hat{C}(\Psi_{11}) = \\ &= a_0\Psi_{00} + a_1\Psi_{01} + a_2\Psi_{11} + a_3\Psi_{10} \end{aligned} \tag{31}$$

If we rewrite (31) in the proper order as

$$\Phi = \rho_0\Psi_{00} + \rho_1\Psi_{01} + \rho_2\Psi_{10} + \rho_3\Psi_{11} \tag{32}$$

the relationship between coefficients $[a_0\ a_1\ a_2\ a_3]$ and $[\rho_0\ \rho_1\ \rho_2\ \rho_3]$ can be expressed in matrix form as

$$\begin{bmatrix} \rho_0 \\ \rho_1 \\ \rho_2 \\ \rho_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} \tag{33}$$

From this, we can conclude that

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{34}$$

is the matrix representation of operator $\hat{C}$ in basis $\{\Psi_{00}, \Psi_{01}, \Psi_{10}, \Psi_{11}\}$.

Remark 1. It is not difficult to verify that $\hat{C}^2 = I$, which implies that the C-NOT operation is reversible.

C-NOT Gates and Bell States

In addition to being a universal building block for quantum computers, the C-NOT gate can also be used to produce *Bell states*, which have the form

$$\Phi_{00} = \frac{1}{\sqrt{2}}(\Psi_{00} + \Psi_{11}) \tag{35}$$

$$\Phi_{01} = \frac{1}{\sqrt{2}} (\Psi_{01} + \Psi_{10}) \tag{36}$$

$$\Phi_{10} = \frac{1}{\sqrt{2}} (\Psi_{00} - \Psi_{11}) \tag{37}$$

$$\Phi_{11} = \frac{1}{\sqrt{2}} (\Psi_{01} - \Psi_{10}) \tag{38}$$

We will now demonstrate how such states can be obtained by combining a C-NOT gate and a Hadamard gate. The circuit in Fig. 2 shows one such configuration, which produces function Φ_{00}.

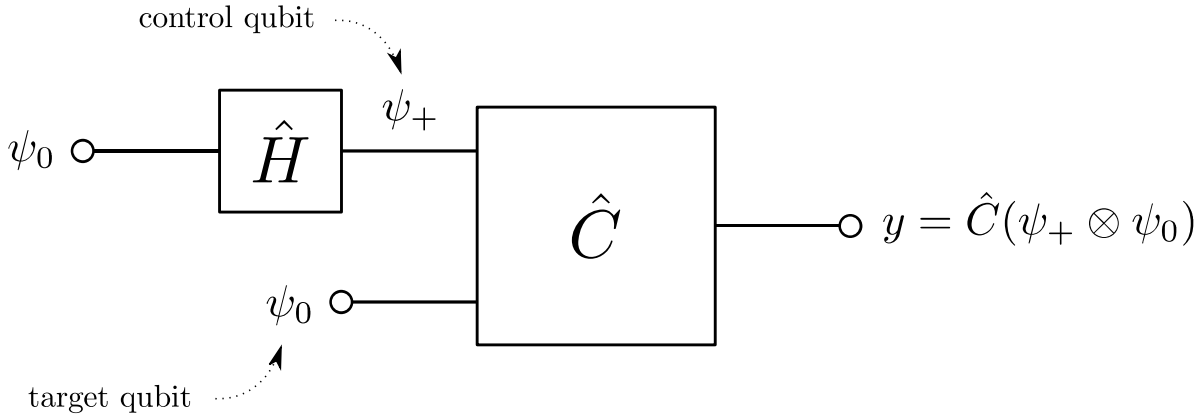

Figure 2: A combination of gates that produces Bell state Φ_{00}.

To explain how this circuit operates, we should first recall that the Hadamard gate generates function

$$\psi_+ = \frac{1}{\sqrt{2}}\psi_0 + \frac{1}{\sqrt{2}}\psi_1 \tag{39}$$

when its input is ψ_0. The C-NOT gate then uses ψ_+ as the *control qubit* and ψ_0 as the *target qubit*, producing

$$\begin{aligned} &\hat{C}\,(\psi_+ \otimes \psi_0) = \frac{1}{\sqrt{2}}\hat{C}\,(\psi_0 \otimes \psi_0) + \frac{1}{\sqrt{2}}\hat{C}\,(\psi_1 \otimes \psi_0) = \\ &= \frac{1}{\sqrt{2}}\hat{C}(\Psi_{00}) + \frac{1}{\sqrt{2}}\hat{C}(\Psi_{10}) = \frac{1}{\sqrt{2}}\,(\Psi_{00} + \Psi_{11}) \end{aligned} \tag{40}$$

This expression obviously corresponds to function Φ_{00}.

We can do something similar for the other three Bell states as well. To realize Bell state Φ_{01}, for example, we simply need to replace the target qubit with ψ_1, in which case we obtain

$$\begin{aligned} &\hat{C}\,(\psi_+ \otimes \psi_1) = \frac{1}{\sqrt{2}}\hat{C}\,(\psi_0 \otimes \psi_1) + \frac{1}{\sqrt{2}}\hat{C}\,(\psi_1 \otimes \psi_1) = \\ &= \frac{1}{\sqrt{2}}\hat{C}(\Psi_{01}) + \frac{1}{\sqrt{2}}\hat{C}(\Psi_{11}) = \frac{1}{\sqrt{2}}\,(\Psi_{01} + \Psi_{10}) = \Phi_{01} \end{aligned} \tag{41}$$

The circuit that accomplishes this is shown in Fig. 3

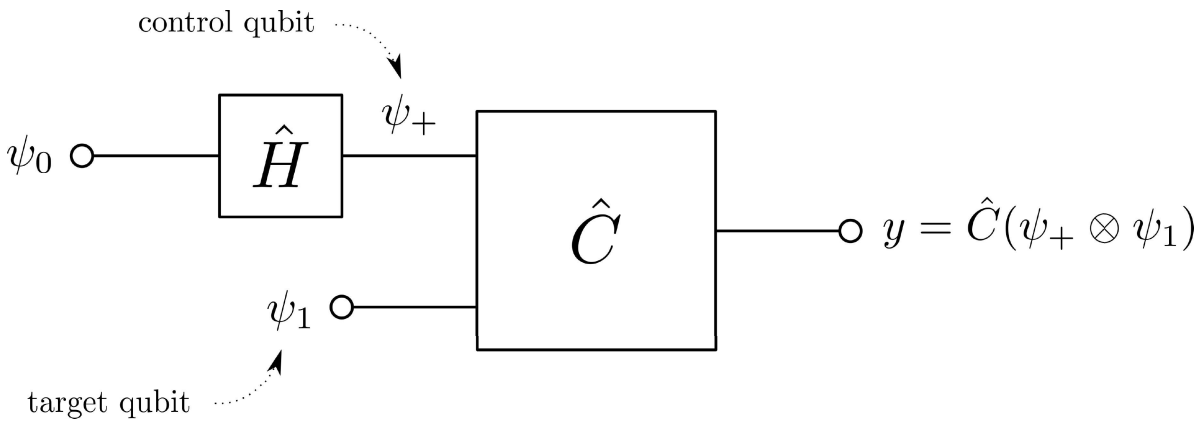

Figure 3: A combination of gates that produces Bell state Φ_{01}.

The Relationship Between C-NOT and XOR Gates

The C-NOT gate is often viewed as a quantum generalization of the classical XOR gate. To see why this is so, we should first recall that the truth table for the standard XOR operation has the form

X	Y	$X \oplus Y$
0	0	0
0	1	1
1	0	1
1	1	0

Table 1. The classical XOR operation.

If we interpret X as the "control" bit and Y as the "target" bit, it is obvious that the output equals the target bit when $X = 0$ and "flips" it when $X = 1$. This is similar to what the quantum C-NOT gate does to the target qubit, but there are several significant differences that we need to take into account. One of them has to do with the fact that an XOR gate always performs a *single* operation on inputs X and Y, producing $X \oplus Y$ as the output. A C-NOT gate, on the other hand, simultaneously operates on *all four* coefficients $[a_0\ a_1\ a_2\ a_3]$ and transforms them into $[\rho_0\ \rho_1\ \rho_2\ \rho_3]$. This suggests that quantum gates have an inherent potential for parallelism that no conventional logic gate can replicate.

Another important difference between C-NOT and XOR gates is that the C-NOT operation is *reversible*, while XOR is *not*. Indeed, given $X \oplus Y = 1$, no subsequent operation on this bit will allow us to recover X and Y (since *two* different combinations correspond to the same output). Something similar can be said for the NAND gate as well, whose output $\overline{X \cdot Y} = 1$ corresponds to *three* possible input combinations. As a result, we are once again unable to determine the original inputs X and Y.

The Quantum Toffoli Gate

Because most conventional logic gates (such as NAND and XOR gates, for example) are *irreversible*, it is impossible to directly map their operation onto a quantum computer. In

order to do that, we would first have to transform the original digital circuit into an equivalent one that consists exclusively of reversible gates. It turns out that this requirement is met if we use so-called *Toffoli gates*, whose schematic description is shown in Fig. 4.

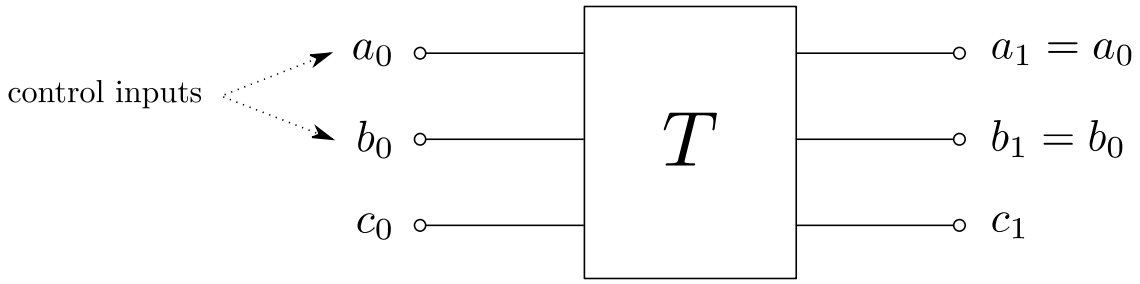

Figure 4: A Toffoli gate.

In this diagram, a_0 and b_0 represent control inputs which are automatically replicated at the output, while c_1 differs from c_0 only when $a_0 = b_0 = 1$. The truth table for such a gate is shown in Table 2, in which the highlighted bits correspond to the two scenarios where $c_1 \neq c_0$.

a_0	b_0	c_0	a_1	b_1	c_1
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	1	0	1
1	1	**0**	1	1	**1**
1	1	**1**	1	1	**0**

Table 2. The truth table for a Toffoli gate.

To see why this operation is reversible, let us consider the cascade connection of two Toffoli gates shown in Fig. 5. The control inputs in the first stage are a_0 and b_0, while a_1 and b_1 assume this role in the second stage. The combined truth table shown in Table 3 indicates that outputs a_2, b_2 and c_2 are identical to a_0, b_0 and c_0, which means that the original inputs can be recovered by performing two consecutive Toffoli operations.

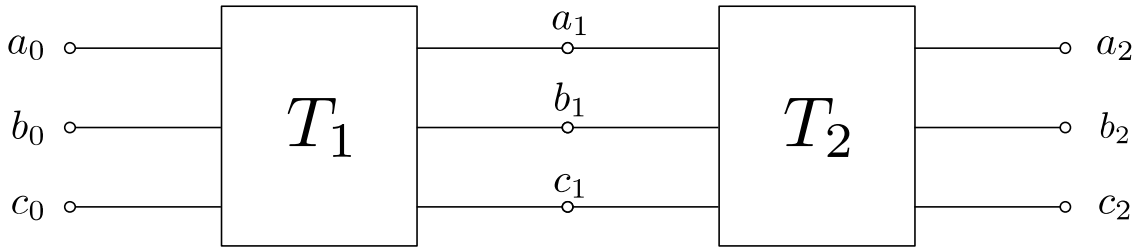

Figure 5: A configuration that performs two consecutive Toffoli operations.

a_0	b_0	c_0	a_1	b_1	c_1	a_2	b_2	c_2
0	0	0	0	0	0	0	0	0
0	0	1	0	0	1	0	0	1
0	1	0	0	1	0	0	1	0
0	1	1	0	1	1	0	1	1
1	0	0	1	0	0	1	0	0
1	0	1	1	0	1	1	0	1
1	1	**0**	1	1	**1**	1	1	**0**
1	1	**1**	1	1	**0**	1	1	**1**

Table 3. Combined truth table for a pair of Toffoli gates.

In order to show that *any* logic circuit can be realized using a combination of Toffoli gates, it suffices to demonstrate that this gate can simulate the NAND operation, since this type of gate is known to be *universal* (i.e., it represents the basic building block from which all other standard gates can be built). The diagram in Fig. 6 shows how this can be done (the possible input and output combinations for this circuit are provided in Table 4). Note that Table 4 includes only half of the input combinations that are associated with a regular Toffoli gate. This is due to the fact that the value of c_0 is fixed at 1.

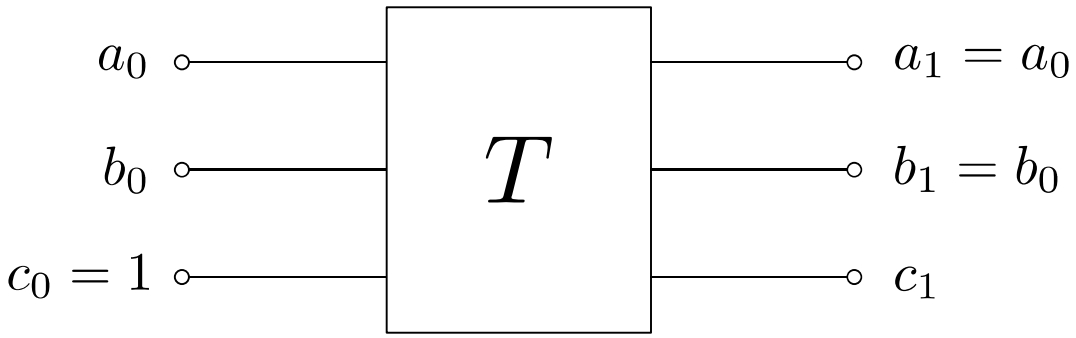

Figure 6: Toffoli gate with input c_0 fixed at 1.

a_0	b_0	c_0	a_1	b_1	c_1
0	0	1	0	0	**1**
0	1	1	0	1	**1**
1	0	1	1	0	**1**
1	1	1	1	1	**0**

Table 4. Truth table for the configuration in Fig. 6.

If we now identify a_0 and b_0 as the inputs and c_1 as the output, the circuit in Fig. 6 can be redrawn in the manner shown in Fig. 7. The truth table that corresponds to this circuit has the form shown in Table 5. It is not difficult to see that this table conforms to the NAND function $c_1 = \overline{a_0 \cdot b_0}$. In view of that, we can think of the configuration shown in Fig. 7 as an equivalent realization of a NAND gate.

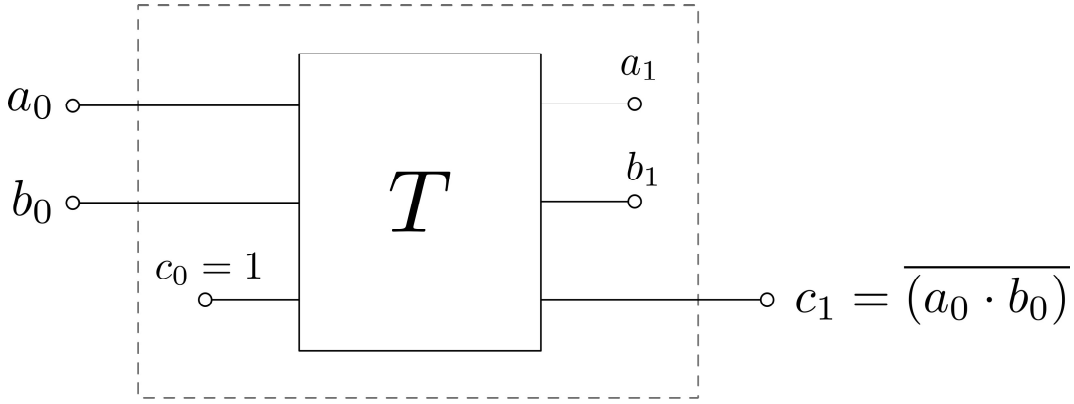

Figure 7: Equivalent realization of a NAND gate.

a_0	b_0	c_1
0	0	**1**
0	1	**1**
1	0	**1**
1	1	**0**

Table 5. Reduced truth table for the configuration in Fig. 6.

What would a *quantum* Toffoli gate look like? In order to explain this, we first need to introduce the Toffoli operator $\hat{T}$, which acts on basis functions

$$\Psi_{ijk} = \psi_i \otimes \psi_j \otimes \psi_k \tag{42}$$

This operator transforms function Ψ_{ijk} into

$$\hat{T}(\Psi_{ijk}) = \Psi_{ij\bar{k}} \tag{43}$$

when $i = j = 1$, and leaves them unchanged otherwise (in expression (43), $\bar{k}$ represents the *complement* of k). Note that this is precisely what the classical Toffoli gate does with inputs a_0, b_0 and c_0 - it leaves outputs a_1 and b_1 unchanged, and c_1 changes only when $a_0 = b_0 = 1$ (in which case it becomes $c_1 = \bar{c}_0$). A schematic diagram of such a gate is shown in Fig. 8.

When operator $\hat{T}$ is applied to a general function of the form

$$\begin{aligned} \Psi = a_0\Psi_{000} + a_1\Psi_{001} + a_2\Psi_{010} + a_3\Psi_{011} + \\ + a_4\Psi_{100} + a_5\Psi_{101} + a_6\Psi_{110} + a_7\Psi_{111} \end{aligned} \tag{44}$$

we obtain

$$\begin{aligned} \Phi = \hat{T}(\Psi) = a_0\Psi_{000} + a_1\Psi_{001} + a_2\Psi_{010} + a_3\Psi_{011} + \\ + a_4\Psi_{100} + a_5\Psi_{101} + a_6\hat{T}(\Psi_{110}) + a_7\hat{T}(\Psi_{111}) \end{aligned} \tag{45}$$

In this expression, operator $\hat{T}$ is shown explicitly only in the last two terms, since it leaves the other ones unchanged. These terms become $\hat{T}(\Psi_{110}) = \Psi_{111}$ and $\hat{T}(\Psi_{111}) = \Psi_{110}$, respectively, which means that coefficients a_6 and a_7 effectively "trade places".

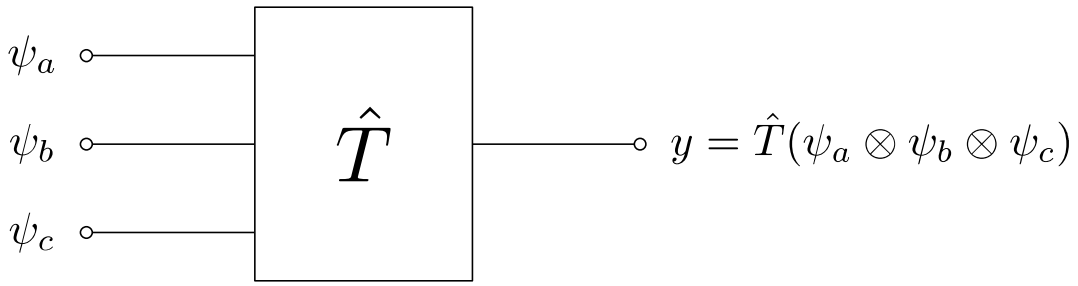

Figure 8: A quantum Toffoli gate.

If we denote the coefficients associated with function Φ in basis $\{\Psi_{000}, \ldots, \Psi_{111}\}$ by $\{\rho_1, \rho_2, \ldots, \rho_7\}$, the relationship between $\{\rho_i\}$ and $\{a_i\}$ can be expressed as

$$\begin{bmatrix} \rho_0 \\ \rho_1 \\ \rho_2 \\ \rho_3 \\ \rho_4 \\ \rho_5 \\ \rho_6 \\ \rho_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{bmatrix} \tag{46}$$

From (46) we can conclude that the matrix representation of $\hat{T}$ in basis $\{\Psi_{000}, \ldots, \Psi_{111}\}$ has the form

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \tag{47}$$

Remark 2. It is not difficult to verify that operator $\hat{T}$ satisfies $\hat{T}^2 = \hat{I}$, which means that a quantum Toffoli gate is *reversible* (just like all the other quantum gates that we considered).

Basic Quantum Algorithms

We will begin by examining two relatively simple quantum algorithms, which were developed by Deutsch and Grover. This will give us an idea of how the quantum approach differs from the conventional one, and what its potential advantages might be. Once we get a "feel" for how these algorithms work, we will consider the so-called *eigenvalue estimation problem*, which is the key for developing an efficient method for prime factorization. This is of fundamental importance, since current encryption schemes rely on the fact that classical computers cannot determine the prime factors of large numbers in a reasonable amount of time.

Deutsch's Algorithm

Deutsch's algoritm is a natural starting point for our discussion, since it is concerned with a simple function $f(x)$ that can take only values 0 or 1. The problem that this algorithm addresses can be envisioned as a series of experiments with a "black box", which takes x as its input and produces $f(x)$ as its output. The function itself is not known to us, but it is available to the "black box" internally (which is what allows it to map x into $f(x)$).

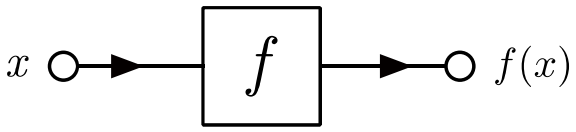

Figure 9: "Black box" for identifying function $f(x)$.

Our goal in the following will be to identify function f by performing a sufficient number of tests on the "black box". We will start by considering the simplest possible scenario, in which the values of x are limited to 0 and 1. Under such circumstances, our objective will be to establish whether or not function f is *constant* (i.e., whether $f(0) = f(1)$). This can be done very easily by evaluating $f(x)$ *twice* (once for $x = 0$ and once for $x = 1$). It turns out, however, that Deutsch's algorithm can accomplish this task even more efficiently, with just a *single* quantum measurement.

To describe how this algorithm works, we will introduce a linear operator $\hat{U}_f$ which acts like a quantum version of the "black box" (in the sense that it can evaluate $f(x)$ internally when presented with function ψ_x). The way this operator works is schematically illustrated in Fig. 10.

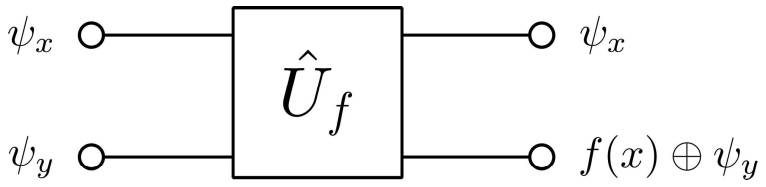

Figure 10: Quantum "black box" for identifying function $f(x)$.

This diagram indicates that the quantum counterpart of the classical "black box" actually has *two* inputs. The first qubit is assumed to be in state ψ_x, and the second one (which

plays an auxiliary role) is assumed to be in state ψ_y. Indices x and y can take values 0 or 1, and $\hat{U}_f$ transforms the tensor product of functions ψ_x and ψ_y into

$$\hat{U}_f(\psi_x \otimes \psi_y) = \psi_x \otimes [f(x) \oplus \psi_y] \tag{48}$$

The symbol $\oplus$ that appears in equation (48) represents an operation that leaves ψ_y *unchanged* if $f(x) = 0$, and "flips" ψ_y to its complementary value $\psi_{\bar{y}}$ if $f(x) = 1$. We can formally describe this relationship as

$$f(x) \oplus \psi_y = \begin{cases} \psi_y, & \text{if } f(x) = 0 \\ \psi_{\bar{y}}, & \text{if } f(x) = 1 \end{cases} \tag{49}$$

where $\bar{y}$ denotes the complement of y.

Since equation (48) tells us how operator $\hat{U}_f$ transforms basis functions $\psi_x \otimes \psi_y$, we can apply it to any function in this space. This pertains to function $\psi_+ \otimes \psi_-$ as well, which can be represented as

$$\begin{aligned} \psi_+ \otimes \psi_- &= \frac{1}{2}(\psi_0 + \psi_1) \otimes (\psi_0 - \psi_1) = \frac{1}{2}(\psi_0 \otimes \psi_0) - \frac{1}{2}(\psi_0 \otimes \psi_1) + \\ &+ \frac{1}{2}(\psi_1 \otimes \psi_0) - \frac{1}{2}(\psi_1 \otimes \psi_1) \end{aligned} \tag{50}$$

Since $\hat{U}_f$ is a linear operator, $\hat{U}_f(\psi_+ \otimes \psi_-)$ becomes

$$\begin{aligned} \hat{U}_f(\psi_+ \otimes \psi_-) &= \frac{1}{2}\hat{U}_f(\psi_0 \otimes \psi_0) - \frac{1}{2}\hat{U}_f(\psi_0 \otimes \psi_1) + \\ &+ \frac{1}{2}\hat{U}_f(\psi_1 \otimes \psi_0) - \frac{1}{2}\hat{U}_f(\psi_1 \otimes \psi_1) \end{aligned} \tag{51}$$

It is hopefully obvious that each of these four terms conforms to the definition provided in (48), and can therefore be evaluated precisely.

Using expression (51) as a starting point, it can be shown that operator $\hat{U}_f$ transforms the tensor product $\psi_+ \otimes \psi_-$ into

$$\hat{U}_f(\psi_+ \otimes \psi_-) = \psi_A \otimes \psi_B \tag{52}$$

where

$$\psi_A = \frac{1}{\sqrt{2}}\psi_0 + \left[(-1)^{f(1)-f(0)}\right] \cdot \frac{1}{\sqrt{2}}\psi_1 \tag{53}$$

and

$$\psi_B = \frac{1}{\sqrt{2}}(-1)^{f(0)}(\psi_0 - \psi_1) = (-1)^{f(0)}\psi_- \tag{54}$$

(see the textbook for a derivation). A schematic description of this relationship is provided in Fig. 11, which indicates that $\hat{U}_f$ places particle 1 in state ψ_A, and particle 2 in state ψ_B.

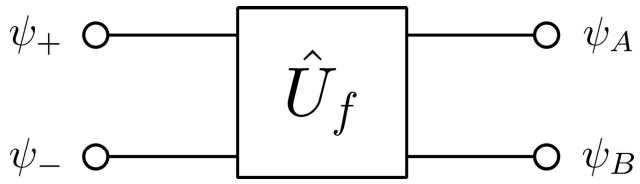

Figure 11: Quantum "black box" with input $\psi_+ \otimes \psi_-$.

How does this transformation help us determine whether function f is constant? To see that, we should first observe that the difference $f(1)-f(0)$ (which appears in the expression for ψ_A) can take the values shown in Table 6.

$f(0)$	$f(1)$	$f(1)-f(0)$
0	0	0
0	1	1
1	0	−1
1	1	0

Table 6. Possible values for $f(1)-f(0)$.

Using this table, it is easily verified that

$$(-1)^{f(1)-f(0)} = \begin{cases} 1, & \text{if } f(0)=f(1) \\ -1, & \text{if } f(0)\neq f(1) \end{cases} \tag{55}$$

If we now apply the Hadamard operator to the first particle and leave the second one intact, we obtain

$$\Psi_{\text{out}} = (\hat{H}\otimes\hat{I})(\psi_A\otimes\psi_B) = (\hat{H}\psi_A)\otimes\psi_B \tag{56}$$

Observing that

$$\begin{aligned} \hat{H}\psi_A &= \frac{1}{\sqrt{2}}\hat{H}\psi_0 + \frac{1}{\sqrt{2}}\left[(-1)^{f(1)-f(0)}\right]\hat{H}\psi_1 = \\ &= \frac{1}{2}(\psi_0+\psi_1) + \frac{1}{\sqrt{2}}\left[(-1)^{f(1)-f(0)}\right]\cdot\frac{1}{\sqrt{2}}(\psi_0-\psi_1) \end{aligned} \tag{57}$$

function $\hat{H}\psi_A$ can be expressed as

$$\hat{H}\psi_A = \frac{1}{2}(\psi_0+\psi_1) + \frac{1}{2}\left[(-1)^{f(1)-f(0)}\right]\cdot(\psi_0-\psi_1) \tag{58}$$

When $f(x)$ is *constant* (i.e., when $f(0)=f(1)$), we know that $(-1)^{f(1)-f(0)}$ equals 1, so $\hat{H}\psi_A$ becomes

$$\hat{H}\psi_A = \frac{1}{2}(\psi_0+\psi_1) + \frac{1}{2}(\psi_0-\psi_1) = \psi_0 \tag{59}$$

If $f(0)\neq f(1)$, expression (55) indicates that

$$(-1)^{f(1)-f(0)} = -1 \tag{60}$$

in which case

$$\hat{H}\psi_A = \frac{1}{2}(\psi_0+\psi_1) - \frac{1}{2}(\psi_0-\psi_1) = \psi_1 \tag{61}$$

The schematic diagram provided in Fig. 12 illustrates how expressions (59) and (61) can help us determine whether or not $f(x)$ is a constant. It shows that we can do so with a *single* measurement on the first particle in basis $\{\psi_0,\psi_1\}$, while any classical algorithm requires *two* separate evaluations of function $f(x)$ (as well as a comparison of the obtained results).

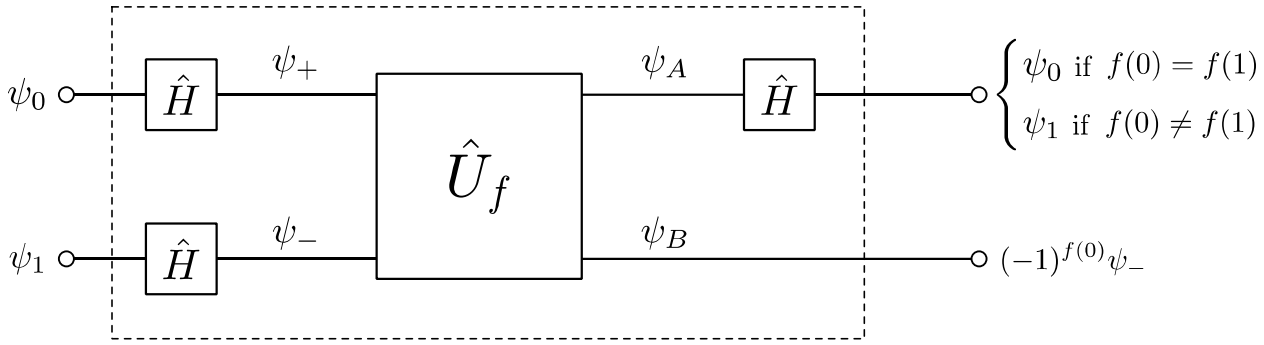

Figure 12: Schematic representation of Deutsch's algorithm.

The Deutsch-Josza Problem

The potential advantages of this approach become even more evident if Deutsch's problem is generalized to include the case when x can take any value in the set $N = \{0, 1, 2, \ldots, 2^n - 1\}$. In this scenario, function $f(x)$ is once again allowed to take only values 0 and 1, but we will additionally assume that it can either be *constant* for all possible x, or can produce an *equal number* of zeros and ones on set N (such a function is said to be *balanced*).

Since x can now take 2^n different values, in the worst case scenario a conventional algorithm would have to check more than half of them before it could establish with certainty whether or not $f(x)$ is constant. To see why this is so, imagine that the first 2^{n-1} tests produced $f(x) = 1$. At this point, it still wouldn't be clear whether $f(x)$ is constant or not, since a balanced function could conceivably have values $f(x) = 1$ for $x \in \{0, 1, \ldots, 2^{n-1} - 1\}$ and $f(x) = 0$ for $x \in \{2^{n-1}, \ldots, 2^n - 1\}$. To rule out this possibility, we would obviously have to perform one more test, which brings the total number of evaluations to $2^{n-1} + 1$.

Since the computational complexity of this process grows exponentially with n, we can safely say that such a task would eventually overwhelm any classical computer, no matter how powerful it is. Even if it were performed in parallel, we would still need a prohibitively large number of processors.

In contrast, a quantum computer with n qubits can determine whether $f(x)$ is constant or balanced from a *single* measurement, using the so-called *Deutsch-Josza algorithm*. One does not need to know the details of this algorithm in order to recognize that it represents a remarkable improvement over the classical approach. The fact that something like this is possible suggests that the quantum paradigm can dramatically speed up certain computational tasks, and can allow us to solve entire classes of problems much more efficiently.

When making such claims, however, it is important to keep in mind that the classical strategy which we described above is *deterministic*, while the quantum one is not. As a result, we should exercise some caution when assessing the actual speed-up that quantum computers can achieve in this case. What is often forgotten in such discussions is that *probabilistic* classical algorithms can solve the Deutsch-Josza problem far more efficiently, and do not require $2^{n-1} + 1$ evaluations. Such algorithms typically compute $f(x)$ for several randomly chosen values of x, and can often establish whether this function is constant or balanced after only a few iterations. If we take that into account, the advantages that the quantum approach offers seem considerably less dramatic.

Grover's Algorithm

Suppose that we have 2^n possible solutions to a problem, and that only one of them is correct. Suppose further that we have numbered these solutions from 0 to $2^n - 1$, and that we have a "black box" which can precisely identify whether or not a solution is correct when we present its integer equivalent (which is denoted by x) at the input.

To make things a little more concrete, let us assume that the correct solution corresponds to $x = C$. This allows us to define function

$$f(x) = \begin{cases} 1 & \text{if } x = C \\ 0 & \text{if } x \neq C \end{cases} \tag{62}$$

which can distinguish correct solutions from incorrect ones. This function is not known to us, but we will assume that the "black box" can evaluate it internally (as was the case with Deutsch's algorithm). Under such circumstances, the "black box" will operate in the manner shown in Fig. 13, producing a 1 only when the correct value of x appears at the input.

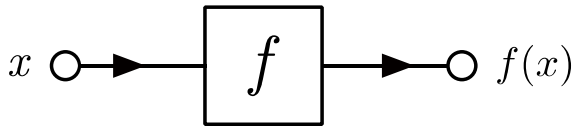

Figure 13: "Black box" for identifying function $f(x)$.

If we were to approach the problem in this way, it could take as many as $2^n - 1$ tests before we find the correct answer, since we would have to examine the various possibilities one by one. It turns out, however, that there is a quantum algorithm which allows us accomplish this task in less than $\sqrt{2^n}$ steps, which can result in a significant reduction in execution time when n is large.

In order to see how this algorithm works, let us consider a system that consists of n qubits. We will assume that its overall state is

$$\Psi = H^{\otimes n}(\Psi_0) = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} \Psi_x \tag{63}$$

where $N = 2^n$, and basis functions Ψ_x represent the possible states that could be observed if we were to measure all n particles simultaneously. In keeping with the notation that we introduced previously, these basis functions should be interpreted as $\Psi_0 = \Psi_{00\ldots0}$, $\Psi_1 = \Psi_{00\ldots1}$, $\ldots$, $\Psi_{N-1} = \Psi_{11\ldots1}$.

Remark 3. We should recall that state Ψ in (63) can be easily obtained using n Hadamard gates, since

$$H^{\otimes n}(\Psi_0) = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} \Psi_x \tag{64}$$

Since we decided to label the "correct" solution by C, we will be interested in the corresponding basis state Ψ_C. This is obviously the state that we would like to observe when we

make a measurement on all n particles, since the binary string that we obtain will correspond to $x = C$. It is not clear how to accomplish that, however, because the probability of each outcome is the *same* when the system is in state Ψ. This follows directly from expression (63), which indicates that

$$P(x) = \left(\frac{1}{\sqrt{N}}\right)^2 = \frac{1}{N} \tag{65}$$

for every choice of x.

The challenge, then, is to determine how this system should be manipulated in order to ensure that Ψ_C becomes the *most likely outcome* when a measurement is made. Grover's algorithm provides an elegant way to do this, and offers a systematic procedure that will allow us to identify the correct solution. We will explain how it works by breaking it down into three simple steps (all intermediate derivations are provided in the textbook).

Step 1 of Grover's Algorithm

We begin by introducing an operator $\hat{U}_f$ that transforms the tensor product of basis state Ψ_x and function ψ_- as

$$\hat{U}_f\left(\Psi_x \otimes \psi_-\right) = (-1)^{f(x)}\Psi_x \otimes \psi_- \tag{66}$$

The function $f(x)$ that appears in (66) is the same one that we introduced in (62) - it equals 1 if $x = C$, and is 0 otherwise.

The effect of this operator is schematically described in Fig. 14. As before, we will assume that the quantum "black box" can internally evaluate $f(x)$ when presented with function Ψ_x at the input. From this figure, we can conclude that $\hat{U}_f$ changes the sign of Ψ_x if x corresponds to the correct solution, and leaves it intact otherwise.

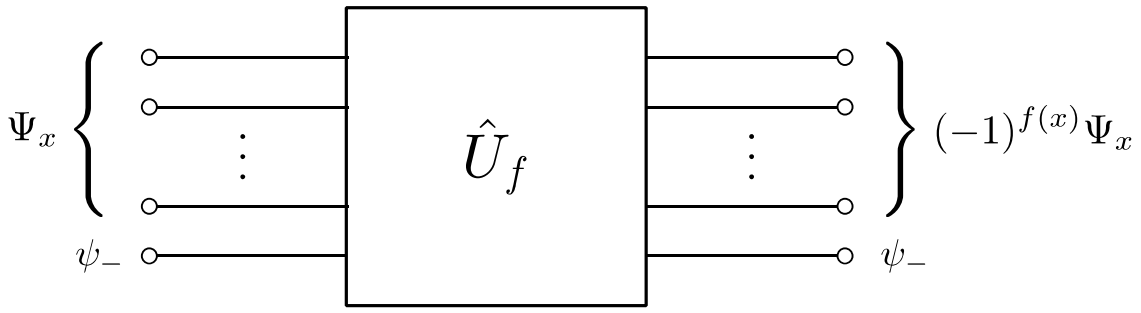

Figure 14: The effect of operator $\hat{U}_f$ on function Ψ_x.

Because ψ_- remains unaffected by this transformation, it is common practice to represent expression (66) as

$$\hat{U}_f\Psi_x = (-1)^{f(x)}\Psi_x \tag{67}$$

This is not entirely correct, of course, since $\hat{U}_f$ operates on the tensor product $\Psi_x \otimes \psi_-$ (and not on Ψ_x itself). We can afford to disregard this discrepancy, however, because it does not affect the final result.

Step 2 of Grover's Algorithm

Let us now return to function Ψ, which represents the input for Grover's algorithm. Since we are interested in identifying component Ψ_C, it will be convenient to represent Ψ as

$$\Psi = \frac{1}{\sqrt{N}}\Psi_C + \frac{1}{\sqrt{N}}\sum_{x\neq C}\Psi_x \tag{68}$$

This expression becomes more compact if we introduce function

$$\Psi_{\bar{C}} = \frac{1}{\sqrt{N}}\sum_{x\neq C}\Psi_x \tag{69}$$

in which case (68) takes the form

$$\Psi = \frac{1}{\sqrt{N}}\Psi_C + \Psi_{\bar{C}} \tag{70}$$

The problem with this representation is that function Ψ_C is normalized (since all basis states satisfy $\langle\Psi_x, \Psi_x\rangle = 1$), but $\Psi_{\bar{C}}$ is *not*. For that reason, it is better to work with function

$$\Psi_W = \sqrt{\frac{N}{N-1}}\,\Psi_{\bar{C}} \tag{71}$$

which differs from $\Psi_{\bar{C}}$ by a constant multiplier. It is not difficult to see that this function satisfies

$$\langle\Psi_W, \Psi_W\rangle = 1 \tag{72}$$

and

$$\langle\Psi_C, \Psi_W\rangle = 0 \tag{73}$$

which makes subsequent derivations easier.

Remark 4. The subscript W in Ψ_W reflects the fact that all components of this function correspond to "wrong" solutions.

Using the notation that we just introduced, we can rewrite Ψ as

$$\Psi = \frac{1}{\sqrt{N}}\Psi_C + \Psi_{\bar{C}} = \frac{1}{\sqrt{N}}\Psi_C + \sqrt{\frac{N-1}{N}}\,\Psi_W \tag{74}$$

For the puroposes of our analysis, it will also be useful to define function

$$\bar{\Psi} = \sqrt{\frac{N-1}{N}}\Psi_C - \frac{1}{\sqrt{N}}\Psi_W \tag{75}$$

which satisfies $\left\langle\Psi, \bar{\Psi}\right\rangle = 0$ and $\left\langle\bar{\Psi}, \bar{\Psi}\right\rangle = 1$ (both of these properties are easily verified). Setting

$$\theta = \sin^{-1}\left(\frac{1}{\sqrt{N}}\right) \quad\Longleftrightarrow\quad \sin\theta = \frac{1}{\sqrt{N}} \tag{76}$$

we can now express functions Ψ and $\bar{\Psi}$ in a more compact form, as

$$\Psi = \frac{1}{\sqrt{N}}\Psi_C + \sqrt{\frac{N-1}{N}}\Psi_W = \sin\theta\,\Psi_C + \cos\theta\,\Psi_W \tag{77}$$

and

$$\bar{\Psi} = \sqrt{\frac{N-1}{N}}\Psi_C - \frac{1}{\sqrt{N}}\Psi_W = \cos\theta\,\Psi_C - \sin\theta\,\Psi_W \tag{78}$$

We will see why it is helpful to work with sines and cosines shortly.

Step 3 of Grover's Algorithm

Our final step will be to introduce an operator $\hat{U}_\Psi$ which produces

$$\hat{U}_\Psi\Psi = \Psi \tag{79}$$

$$\hat{U}_\Psi\bar{\Psi} = -\bar{\Psi} \tag{80}$$

when applied to functions Ψ and $\bar{\Psi}$. Such an operator is not difficult to construct, and satisfies

$$\hat{U}_\Psi(\hat{U}_f\Psi) = \sin 3\theta\,\Psi_C + \cos 3\theta\,\Psi_W \tag{81}$$

If we apply this pair of operators k times, we obtain

$$(\hat{U}_\Psi\hat{U}_f)^k\Psi = \sin(2k+1)\theta\,\Psi_C + \cos(2k+1)\theta\,\Psi_W \tag{82}$$

(a derivation of this result is provided in the textbook).

To see why expression (82) is useful, we should recall that

$$\Psi_W = \sqrt{\frac{N}{N-1}}\,\Psi_{\bar{C}} = \sqrt{\frac{N}{N-1}}\cdot\frac{1}{\sqrt{N}}\sum_{x\neq C}\Psi_x = \frac{1}{\sqrt{N-1}}\sum_{x\neq C}\Psi_x \tag{83}$$

Substituting (83) into (82) allows us to rewrite this expression as

$$(\hat{U}_\Psi\hat{U}_f)^k\Psi = [\sin(2k+1)\theta]\;\Psi_C + \sum_{x\neq C}\left[\frac{\cos(2k+1)\theta}{\sqrt{N-1}}\right]\Psi_x \tag{84}$$

Since functions $\{\Psi_0, \Psi_1, \ldots, \Psi_{N-1}\}$ constitute an orthonormal basis, the coefficients next to them can be associated with probabilities. This allow us to conclude that the probability of observing state Ψ_C is

$$P(\Psi_C) = |\sin(2k+1)\theta|^2 \tag{85}$$

when we measure all n particles simultaneously. Any other outcome (which necessarily corresponds to $x \neq C$) has probability

$$P(\Psi_x) = \frac{|\cos(2k+1)\theta|^2}{N-1} \tag{86}$$

Since our objective is to maximize the likelihood that we will record state Ψ_C when we perform a measurement, it would be desirable to pick k in such a way that

$$\sin(2k+1)\theta = 1 \tag{87}$$

is satisfied. This will obviously be the case if $(2k+1)\theta = \pi/2$, which means that k should be chosen as

$$k = \frac{\pi}{4\theta} - \frac{1}{2} \tag{88}$$

Recalling that

$$\sin\theta = \frac{1}{\sqrt{N}} \tag{89}$$

and that $\sin\theta \approx \theta$ when θ is small (which is the case if N is a large number), we can approximate θ as

$$\theta \approx \frac{1}{\sqrt{N}} \tag{90}$$

Substituting (90) into (88), it follows that the optimal value for k is

$$k = \frac{\pi}{4}\sqrt{N} - \frac{1}{2} = \frac{\pi}{4}\sqrt{2^n} - \frac{1}{2} \tag{91}$$

Observing that

$$\frac{\pi}{4}\sqrt{2^n} - \frac{1}{2} < \sqrt{2^n} \tag{92}$$

it follows that Grover's algorithm requires fewer than $\sqrt{2^n}$ steps.

When interpreting expression (91), it is important to keep in mind that this condition usually cannot be satisfied exactly, since k must be an integer. Nevertheless, we can claim that the probability of observing state Ψ_C (and therefore of finding the correct solution) will be very high if we round k off to the closest whole number. To illustrate this point, suppose that $n = 20$, in which case $N = 2^n = 1,048,576$. Equation (91) will then produce $k = 803.7477$, which which suggests that we should choose $k = 804$. If we do so, the probability of registering Ψ_C will be

$$P = \left|\sin\left[(2k+1)\theta\right]\right|^2 = 0.999999756965361 \tag{93}$$

so it is fair to say that such an outcome is virtually certain.

Remark 5. Note that the number of steps needed to identify the correct solution is $1,304$ times smaller than the worst case scenario for the classical approach (remember that this approach could require as many as $2^n - 1 = 1,048,575$ steps if we are really, really unlucky).