Basic Quantum Algorithms

Renato Portugal

Full Researcher at the
National Laboratory of Scientific Computing
LNCC/MCTI

September 27, 2022

Abstract

Quantum computing is evolving so quickly that forces us to revisit, rewrite, and update the basis of the theory. Basic Quantum Algorithms revisits the first quantum algorithms. It started in 1985 with Deutsch trying to evaluate a function at two domain points simultaneously. Then, Deutsch and Jozsa created in 1992 a quantum algorithm that determines whether a Boolean function is constant or balanced. In the following year, Bernstein and Vazirani realized that the same algorithm can be used to find a specific Boolean function in the set of linear Boolean functions. In 1994, Simon presented a new quantum algorithm that determines whether a function is one-to-one or two-to-one exponentially faster than any classical algorithm for the same problem. In the same year, Shor created two new quantum algorithms for factoring integers and calculating discrete logarithms, threatening the cryptography methods widely used nowadays. In 1995, Kitaev described an alternative version for Shor's algorithms that proved useful in many other applications. In the following year, Grover created a quantum search algorithm quadratically faster than its classical counterpart. In this work, all those remarkable algorithms are described in detail with a focus on the circuit model.

Contents

Chapter 1

Introduction

Quantum algorithm is a subarea of quantum computing that is evolving quickly not only in terms of new algorithms but also in terms of applications and implementations. The basic algorithms are the pillars of this new edifice. The construction started with a change in the rules of the game. Instead of storing information in bits, which take either zero or one, we are allowed to store information in qubits, the state of which is a superposition of zeroes and ones. The rules based on classical mechanics were replaced by rules based on quantum mechanics.

The first insight came with Deutsch's proposal in 1985 of evaluating a one-bit Boolean function at two points simultaneously by exploiting the superposition of zeroes and ones, which is known as quantum parallelism. At this point in time, a framework for creating new algorithms was missing, which was only established in 1989 when Deutsch published a paper on quantum circuits, introducing the quantum gates that take place of the well-known classical gates, such as AND, OR, NOT. In 1992, the area of quantum algorithms gained momentum when Deutsch and Jozsa created an algorithm to determine whether a Boolean function is balanced or constant. This algorithm stimulated the development of oracle-based algorithms. We have a function at our disposal and we need to reveal a hidden property of this function. We can evaluate the function as many times as we want, but the goal is to find the property by querying as few as possible.

Bernstein and Vazirani noted in 1993 that the Deutsch-Jozsa algorithm could be used to find a specific Boolean function in the set of linear Boolean functions. The Bernstein-Vazirani algorithm is faster than its classical counterpart without exploiting entanglement. The power of the algorithm relies only on quantum parallelism.

The momentum increased even further when Simon in 1994 published a quantum algorithm that distinguishes whether a function is one-to-one or two-to-one exponentially faster than classical algorithms for the same purpose. In this case, we have to extend the function type in order to have an output with many bits. This algorithm exploits maximum entanglement. Simon's problem has an interesting application to finding a hidden subgroup of a special class of groups.

Shor reached the summit when he created in 1994 two celebrated quantum algorithms for factoring composite integers and calculating discrete logarithms, which have a strong impact in breaking cryptography methods used in our daily life. Shor's algorithms helped to bring quantum computing into the spotlight. Since then, the area is increasing at an amazing rate. Shor's algorithm can also be formulated as an oracle-based algorithm with a function that is periodic. The goal is to find the period by evaluating the function as few as possible. To find periods is a job for the Fourier transform, which is useful in classical algorithms despite

being $O(N \log N)$, where N is the data size. In the quantum case, the Fourier transform is implemented using $O(\log^2 N)$ universal gates and the quantum superposition makes the magic.

Kitaev presented in 1995 another version of Shor's algorithms after developing a quantum algorithm for phase estimation. Given a unitary operator and one of its eigenvectors, the algorithm efficiently finds the corresponding eigenvalue, which is totally characterized by its phase. Kitaev's algorithm proved useful for other applications such as quantum counting.

Grover put the focus on unsorted databases and in 1996 created a quantum algorithm that finds an item quadratically faster than classical searching. Grover's algorithm can also be formulated as an oracle-based algorithm with a Boolean function that is constant except for a single point in the domain. The goal is to find that point by evaluating the function as few as possible. When it is written as a black-box algorithm, it is clear that Grover's algorithm has wide applicability.

Basic Quantum Algorithms describes in full detail the remarkable contributions outlined above. There is no hope if we don't use the correct language: Mathematics, more specifically, linear algebra. Notions such as quantum superposition and entanglement have a precise definition when we use linear algebra. Measurements are described by projectors, gates by unitary operators, and qubits by vectors. Projectors, unitary operators, and vectors are words of the linear algebra language. When describing quantum algorithms or anything related to quantum, it is better to use mathematics because without it someone will probably utter nonsense.

Basic Quantum Algorithms follows as much as possible the historical ordering, which is the order of increasing complexity. We feel like climbing steps with increasing heights, strengthening muscles, and preparing for the challenge of understanding complex quantum algorithms. Each Chapter is as independent as possible to help readers that are familiar with some algorithms skip some parts.

Last but not least, do not hesitate in contacting the author (`portugal@lncc.br`) if there are errors or any problems in terms of imprecision or missing citations. Suggestions are also welcome.

Acknowledgments

The author thanks P.H.G. Lugão for suggesting improvements.

Chapter 2

Quantum Circuits

The goal of this Chapter is to define the concepts of qubit, logic gate, and quantum circuit. Before that, we briefly review key facts of linear algebra [2, 64] using the Dirac notation from the beginning. Part of this material was published in the tutorial [50]. References for this Section are [54] and [67]. Additional references for linear algebra for quantum computing are Appendix A of [49] and Section 2.1 of [45].

2.1 Review of linear algebra using the Dirac notation

There are several notations to show that a variable v is a vector, for example, $\vec{v}$, $\mathbf{v}$, and so on. In quantum computing, the most used is Dirac's: $|v\rangle$. A sequence of vectors is denoted by $|v_0\rangle$, $|v_1\rangle$, $|v_2\rangle$ and so on. It is very common to abuse this notation and denote the same sequence as $|0\rangle$, $|1\rangle$, $|2\rangle$ and so on.

The canonical basis of a two-dimensional vector space has two vectors, denoted by $\{|0\rangle, |1\rangle\}$ in the Dirac notation, where $|0\rangle$ and $|1\rangle$ have the following representation

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad \text{and} \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

These vectors have two entries or components, unit length, and are orthogonal. Then, this basis is orthonormal. It is called *canonical basis* in linear algebra and *computational basis* in quantum computing. Note that $|0\rangle$ is not the null vector, but the first vector of the canonical basis. All entries of the null vector are equal to 0. In the two-dimensional case, it is

$$\begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

without any special denomination in the Dirac notation.

A generic vector in a two-dimensional vector space is obtained via the *linear combination* of the basis vectors,

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle,$$

where α and β are complex numbers. These numbers are the entries of vector $|\psi\rangle$, as can be seen from the notation

$$|\psi\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix}.$$

The *dual vector* to vector $|\psi\rangle$ is denoted by $\langle\psi|$ and is obtained by transposing $|\psi\rangle$ and conjugating each entry. Using the previous equation, we obtain

$$\langle\psi| = \begin{pmatrix}\alpha^* & \beta^*\end{pmatrix},$$

which can be written as

$$\langle\psi| = \alpha^*\langle 0| + \beta^*\langle 1|,$$

where

$$\langle 0| = \begin{pmatrix}1 & 0\end{pmatrix} \quad \text{and} \quad \langle 1| = \begin{pmatrix}0 & 1\end{pmatrix}.$$

The dual vector $\langle\psi|$ is an 1×2 matrix and vector $|\psi\rangle$ is a 2×1 matrix. At this point, we introduce the *dagger* symbol, denoted by †, which is the notation for the conjugate transpose vector (transpose the vector and then conjugate each entry or vice versa). Then, we may write $\langle\psi| = |\psi\rangle^\dagger$ and $|\psi\rangle = \langle\psi|^\dagger$. Two daggers one after the other work as the identity operation.

Suppose that $|\psi_1\rangle$ and $|\psi_2\rangle$ are two-dimensional vectors given by

$$|\psi_1\rangle = \begin{pmatrix}\alpha\\ \beta\end{pmatrix} \quad \text{and} \quad |\psi_2\rangle = \begin{pmatrix}\gamma\\ \delta\end{pmatrix}.$$

The *inner product* of two vectors $|\psi_1\rangle$ and $|\psi_2\rangle$ is a complex number denoted by $\langle\psi_1|\psi_2\rangle$ and defined as the matrix product of the dual vector $\langle\psi_1|$ by $|\psi_2\rangle$, as follows

$$\langle\psi_1|\psi_2\rangle = \begin{pmatrix}\alpha^* & \beta^*\end{pmatrix}\begin{pmatrix}\gamma\\ \delta\end{pmatrix} = \alpha^*\gamma + \beta^*\delta.$$

In the Dirac notation, the calculation of the inner product is performed by distributing the matrix product over the sum of vectors, as follows

$$\langle\psi_1|\psi_2\rangle = \big(\alpha^*\langle 0| + \beta^*\langle 1|\big)\cdot\big(\gamma|0\rangle + \delta|1\rangle\big) = \alpha^*\gamma\,\langle 0|0\rangle + \beta^*\delta\,\langle 1|1\rangle = \alpha^*\gamma + \beta^*\delta.$$

The *norm* of vector $|\psi_1\rangle$ is denoted by $\|\,|\psi_1\rangle\,\|$ and defined as

$$\|\,|\psi_1\rangle\,\| = \sqrt{\langle\psi_1|\psi_1\rangle} = \sqrt{|\alpha|^2 + |\beta|^2},$$

where $|\alpha|$ is the *absolute value* of α, that is

$$|\alpha| = \sqrt{\alpha\cdot\alpha^*}.$$

If $\alpha = a + b\,\mathrm{i}$, where i is the imaginary unit ($\mathrm{i} = \sqrt{-1}$), a is the real part and b is the imaginary part, then

$$|\alpha| = \sqrt{(a + b\,\mathrm{i})\cdot(a - b\,\mathrm{i})} = \sqrt{a^2 + b^2}.$$

A complex number α such that $|\alpha| = 1$ is called a *unit complex number* and can be written as $\mathrm{e}^{\mathrm{i}\theta} = \cos\theta + \mathrm{i}\sin\theta$, where θ is an angle. In real vector spaces, the inner product is called *scalar product* and is given by

$$\langle\psi_1|\psi_2\rangle = \|\,|\psi_1\rangle\|\;\|\,|\psi_2\rangle\|\cos\theta,$$

where θ is the angle between vectors $|\psi_1\rangle$ and $|\psi_2\rangle$.

Using these definitions, we can show that the basis $\{|0\rangle, |1\rangle\}$ is orthonormal, that is, vectors $|0\rangle$ and $|1\rangle$ are orthogonal and have norm 1, that is

$$\langle 0|0\rangle = 1, \quad \langle 0|1\rangle = 0, \quad \langle 1|0\rangle = 0, \quad \langle 1|1\rangle = 1.$$

An algebraic way of denoting orthonormality and of compacting the last four equations into one is

$$\langle k|\ell\rangle = \delta_{k\ell},$$

where k and ℓ are bits and $\delta_{k\ell}$ is the *Kronecker delta*, defined as

$$\delta_{k\ell} = \begin{cases} 1, & \text{if } k = \ell, \\ 0, & \text{if } k \neq \ell. \end{cases}$$

2.2 Qubit and superposition

The basic memory unit of a classical computer is the *bit*, which assumes 0 or 1. Usually, the bit is implemented using two distinct voltages, following the convention that null or low voltage represents bit 0 and high voltage represents bit 1. To determine whether the output is bit 0 or 1 at the end of the computation, it is necessary to measure the voltage.

The basic memory unit of a quantum computer is the *qubit*, which also assumes, at the end of the computation, 0 or 1. The qubit is implemented using an electric current in a small superconductor, following the convention that clockwise current represents 0 and counter-clockwise current represents 1, or the other way around. The difference from the classical device happens during the computation since the qubit admits the simultaneous coexistence of 0 and 1. During the computation, that is, before the measurement, the *state* of a qubit is represented by a norm-1 two-dimensional vector and the states of a qubit corresponding to 0 and 1 are $|0\rangle$ and $|1\rangle$. The definition of *state* is a vector of norm 1 in a complex vector space endowed with the inner product presented in the previous Section.[1] The state can be thought of as the "value" of the qubit before the measurement. Quantum coexistence is represented mathematically by a linear combination of orthonormal vectors as follows

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle,$$

where α and β are complex numbers that obey the constraint

$$|\alpha|^2 + |\beta|^2 = 1.$$

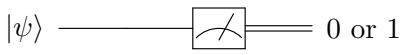
The state of the qubit is vector $|\psi\rangle$ of norm 1 with the entries α and β. The complex numbers α and β are the *amplitudes* of the state $|\psi\rangle$.

The coexistence of bits 0 and 1 cannot be implemented in a classical device, since it is not possible to have low and high voltage at the same time, as everyone knows. In quantum mechanics, it is hard to believe, it is possible to have a quantum system (usually microscopic) with low and high voltage at the same time. This coexistence can only be fully maintained if the quantum system is fully isolated from the surrounding macroscopic environment. When we measure the quantum system to determine the value of the voltage, the measuring device inevitably affects

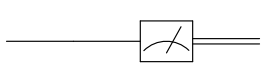
[1] A finite-dimensional vector space with an inner product is a *Hilbert space*.

the voltage, outputting a stochastic result, which is a low or high voltage, similar to the classical bit. In other words, coexistence is only maintained when no one (and no thing) is trying to determine whether the voltage is high or low. Note that quantum mechanics is a *scientific theory*, that is, its laws and results can be tested objectively in laboratories. In addition, unnecessary laws and statements are summarily discarded. Therefore, the statement "coexistence is only maintained when the system is isolated" has practical consequences and is a statement that has been tested and re-tested for over 100 years in thousands of quantum mechanics laboratories worldwide. On the other hand, alternative theories that are more classically palatable, which could help in understanding or visualizing the quantum superposition, have been ruled out by experimental tests.

From a computational point of view, we have a qubit in superposition and we use this feature in a circuit. For example, the circuit



tells us that the initial "value" of the qubit is $|\psi\rangle$ and this information is conveyed unchanged from left to right until a measurement is performed, as shown by the *meter* (the display of a voltmeter). The measurement outputs 0 or 1. Classical information is conveyed by a double wire. If the state of the qubit is $|\psi\rangle = |0\rangle$, a measurement will necessarily output 0 and if the state is $|1\rangle$, a measurement will necessarily output 1. In the general case, if the state is $\alpha|0\rangle + \beta|1\rangle$, a measurement will return 0 with probability $|\alpha|^2$ or 1 with probability $|\beta|^2$, as shown in the circuit



$$\begin{cases} 0, \text{ with probability } |\alpha|^2, \\ 1, \text{ with probability } |\beta|^2. \end{cases}$$

The output can be depicted by a histogram of the probability distribution. It is important to repeat the fact that α and β are called *amplitudes* of the state $\alpha|0\rangle + \beta|1\rangle$ and are numbers that can be negative and may have an imaginary part. On the other hand, $|\alpha|^2$ and $|\beta|^2$ are positive real numbers in the interval $[0, 1]$ and are called probabilities. A careless interchange between amplitudes and probabilities creates unforgivable errors.

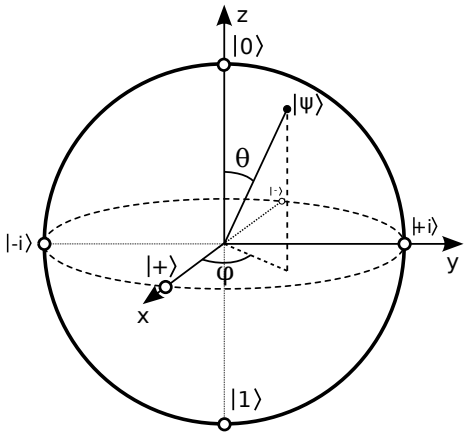


Figure 2.1: Bloch sphere and the location of states $|0\rangle$, $|1\rangle$, $|\pm\rangle$, and $|\pm i\rangle$. An arbitrary state $|\psi\rangle$ is shown with spherical angles θ and φ.

The state of a qubit can be characterized by two angles θ and φ as follows

$$|\psi\rangle = \cos\frac{\theta}{2}|0\rangle + \mathrm{e}^{\mathrm{i}\varphi}\sin\frac{\theta}{2}|1\rangle,$$

where $0 \le \theta \le \pi$ and $0 \le \varphi < 2\pi$. This notation shows that there is a one-to-one correspondence between the set of states of a qubit and points on the surface of a sphere of radius 1, called *Bloch sphere*. The angles θ and φ are spherical angles that describe the location of the state $|\psi\rangle$, as shown in Fig. 2.1. A point on the Bloch sphere is described by a three-dimensional vector with real entries

$$\begin{pmatrix} \sin\theta\cos\varphi \\ \sin\theta\sin\varphi \\ \cos\theta \end{pmatrix}.$$

The locations on the Bloch sphere of states $|0\rangle$ and $|1\rangle$, whose spherical angles are $(\theta,\varphi) = (0,0)$ and $(\theta,\varphi) = (\pi,0)$, are pinpointed in Fig. 2.1. The locations of states

$$\begin{aligned} |\pm\rangle &= \frac{|0\rangle \pm |1\rangle}{\sqrt{2}}, \\ |\pm\mathrm{i}\rangle &= \frac{|0\rangle \pm \mathrm{i}|1\rangle}{\sqrt{2}} \end{aligned}$$

are determined after figuring out the spherical angles, which are $(\theta,\varphi) = (\pi/2, \pi/2 \mp \pi/2)$ and $(\theta,\varphi) = (\pi/2, \pm\pi/2)$, respectively.[2] Then, their locations are the intersections of the x-axis with the Bloch sphere, and y-axis with the Bloch sphere.

If we have an arbitrary 1-qubit state $\alpha|0\rangle + \beta|1\rangle$ and we want to find the spherical angles θ and φ, the first thing to do is to write α and β as $r_1\mathrm{e}^{\mathrm{i}\varphi_1}$ and $r_2\mathrm{e}^{\mathrm{i}\varphi_2}$, respectively, where $r_1 = |\alpha|$ and $r_2 = |\beta|$. Now we multiply the state by $\mathrm{e}^{-\mathrm{i}\varphi_1}$ to obtain $r_1|0\rangle + \mathrm{e}^{\mathrm{i}(\varphi_2-\varphi_1)}r_2|1\rangle$. Then, we take $\varphi = \varphi_2 - \varphi_1$ and $\theta = 2\arccos r_1$. Note that $r_2 = \sin(\theta/2)$ because $r_1^2 + r_2^2 = 1$. There is no problem in multiplying the state by a unit complex number such as $\mathrm{e}^{-\mathrm{i}\varphi_1}$ because in quantum mechanics two quantum states that differ by a global factor are considered equivalent and have the same location on the Bloch sphere. The global factor must be a unit complex number and is usually called *global phase factor*.

2.3 Single-qubit gates

A single-qubit gate is a two-dimensional unitary square matrix. A matrix U is unitary if multiplying U by an arbitrary norm-1 vector results in a norm-1 vector. Formally, suppose that $|\psi'\rangle = U|\psi\rangle$, where $|\psi\rangle$ is a norm-1 two-dimensional vector. If U is a unitary matrix, then $|\psi'\rangle$ will have norm 1. For example, the Hadamard matrix

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

is unitary. Therefore, the multiplication of H by the basis vectors has to result in norm-1 vectors. In fact,

$$H|0\rangle = H\begin{pmatrix}1\\0\end{pmatrix} = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\1\end{pmatrix} = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle.$$

[2] Do not try to understand $\pm$ simultaneously because our mind enters in undesired, superposed, entangled states. Please address first the upper sign of all expressions. Then, address the lower sign.

Note that the resulting vector has norm 1. We denote this vector by $|+\rangle$, that is

$$|+\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle.$$

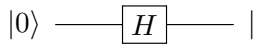
Multiplying H by $|1\rangle$ yields vector $|-\rangle$ defined as

$$|-\rangle = \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle,$$

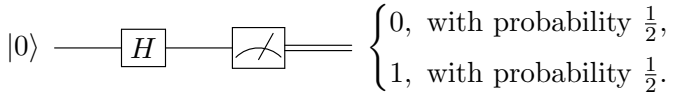
which also has norm 1. These calculations are important because we need to learn the output of the gate. If the input is $|0\rangle$ then the output is $|+\rangle$. If the input is $|1\rangle$, the output is $|-\rangle$. If the input is a superposition $\alpha|0\rangle + \beta|1\rangle$, the output is the superposition of $|+\rangle$ and $|-\rangle$ with the same amplitudes, $\alpha|+\rangle + \beta|-\rangle$, because we use the linearity property of the gate, that is, instead of thinking that H is a matrix, we use that H is a linear operator and if H is applied to a linear combination of vectors $|0\rangle$ and $|1\rangle$ with amplitudes α and β, the result is a linear combination of $H|0\rangle$ and $H|1\rangle$ with the same amplitudes α and β. We can avoid this abstract point of view, but when we multiply a matrix by a sum of vectors, we have to distribute the multiplication over the vectors.

To check that the multiplication of H by the vectors of an orthonormal basis results in norm-1 vectors is not enough to show that H is a unitary matrix. It is also necessary to show that the resulting vectors are orthogonal, that is, to show that $\langle -|+\rangle = 0$. At this point, it is easier to check that H is unitary by calculating $HH^\dagger$, where $H^\dagger$ is obtained by transposing H and conjugating each entry. If the result is the identity matrix, H is unitary. For the record, $H^\dagger$ is called the *Hermitian transpose* of H.

A *quantum circuit* is a graphic representation of a quantum algorithm. The input qubit is on the left and the information (qubit's state) is conveyed unchanged from left to right until it faces a logical gate. The gate receives the input from the left, then processes the information, and the result exits to the right and continues its course. The gate processing is done by multiplying the unitary matrix that represents the gate by the vector that represents the state of the qubit. For example, the expression $|+\rangle = H|0\rangle$ is represented by the following circuit:



The input is vector $|0\rangle$, which is conveyed unchanged by the wire to H, which acts on the input and transforms it into $|+\rangle$, which is then conveyed to the right. The gate action is calculated by multiplying H by $|0\rangle$. Therefore, the result of the computation is $|+\rangle$. If at the end of the computation we perform a measurement, the circuit is



The circuit shows that the output of the measurement of the qubit, whose state was $|+\rangle$, is 0 with probability 1/2 or 1 with the same probability. Fig. 2.2 shows the histogram of the probability distribution generated in Qiskit[3] with two iterations.

[3]Qiskit is open-source software for running programs on IBM quantum computers.

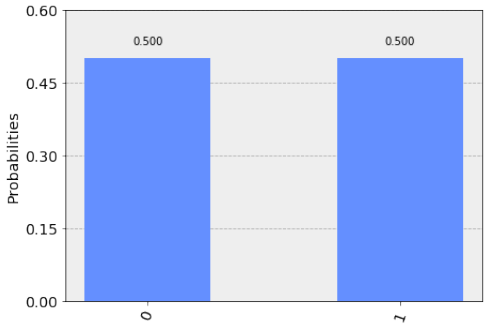


Figure 2.2: Histogram of the probability distribution generated by measuring a qubit whose state is $|+\rangle$.

An example that is simpler than the previous one is the X gate, defined as

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

X is the quantum NOT gate because $|1\rangle = X|0\rangle$ and $|0\rangle = X|1\rangle$. We can verify these equations through the matrix multiplication of X with $|0\rangle$ and $|1\rangle$. In a more compact form, we can write $|j \oplus 1\rangle = X|j\rangle$, where $\oplus$ is the XOR operation or sum modulo 2. Because of this, the gate X is also represented as $\oplus$. A circuit using the X gate is

$|0\rangle$ —[X]— [meter] ═ 1 with probability 1.

Now we can increase the complexity. How to generate a superposition such that the amplitudes of $\alpha|0\rangle + \beta|1\rangle$ are different and non-null? For example, how to generate a state $\alpha|0\rangle + \beta|1\rangle$ such that $|\alpha|^2 = 25\%$ and $|\beta|^2 = 75\%$ before the measurement? The answer is to use the most general 1-qubit gate, whose algebraic expression is

$$U(\theta, \phi, \lambda) = \begin{bmatrix} \cos\frac{\theta}{2} & -e^{i\lambda}\sin\frac{\theta}{2} \\ e^{i\phi}\sin\frac{\theta}{2} & e^{i(\lambda+\phi)}\cos\frac{\theta}{2} \end{bmatrix}.$$

After applying $U(\theta, 0, 0)$ on $|0\rangle$, we obtain

$$U(\theta, 0, 0)|0\rangle = \cos\frac{\theta}{2}|0\rangle + \sin\frac{\theta}{2}|1\rangle.$$

We must choose $\theta = 2\pi/3$, because $\cos^2(\pi/3) = 1/4$. The previous example using the Hadamard gate can be reproduced by taking $\theta = \pi/2$ and $\lambda = \pi$ because $H = U(\pi/2, 0, \pi)$.

$U(\theta, \phi, \lambda)$ is a wildcard gate because it represents all 1-qubit gates (modulo a global phase factor). For instance, three useful gates obtained from U are

$$U\left(\theta, -\frac{\pi}{2}, \frac{\pi}{2}\right) = R_x(\theta) = \begin{bmatrix} \cos\frac{\theta}{2} & -i\sin\frac{\theta}{2} \\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{bmatrix},$$

$$U(\theta, 0, 0) = R_y(\theta) = \begin{bmatrix} \cos\frac{\theta}{2} & -\sin\frac{\theta}{2} \\ \sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{bmatrix},$$

$$U(0, 0, \lambda) = e^{i\lambda/2}R_z(\lambda) = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\lambda} \end{bmatrix}.$$

where R_x, R_y, and R_z are the operators that rotate the Bloch sphere about the x, y, and z-axis, respectively. Unfortunately, U is too perfect to be true for it would be possible to choose θ as small as we want at the cost of one single gate. Errors would eventually prevent obtaining good results after choosing very small angles.

To determine the computational complexity of an algorithm, we have to restrict ourselves to a finite set of single-qubit gates. The most important 1-qubit gates are

$$I_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad Y = \begin{bmatrix} 0 & -\mathrm{i} \\ \mathrm{i} & 0 \end{bmatrix}, \quad Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

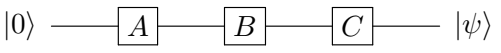
known as *Pauli matrices*,

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \quad S = \begin{bmatrix} 1 & 0 \\ 0 & \mathrm{i} \end{bmatrix}, \quad S^\dagger = \begin{bmatrix} 1 & 0 \\ 0 & -\mathrm{i} \end{bmatrix}, \quad T = \begin{bmatrix} 1 & 0 \\ 0 & \mathrm{e}^{\frac{\mathrm{i}\pi}{4}} \end{bmatrix}, \quad T^\dagger = \begin{bmatrix} 1 & 0 \\ 0 & \mathrm{e}^{-\frac{\mathrm{i}\pi}{4}} \end{bmatrix}$$

known as Hadamard gate, phase gate, its conjugate gate, $\pi/8$ gate or T gate and its conjugate.[4] The complex numbers $\mathrm{e}^{\pm\frac{\mathrm{i}\pi}{4}}$ are equal to

$$\mathrm{e}^{\pm\frac{\mathrm{i}\pi}{4}} = \frac{1 \pm \mathrm{i}}{\sqrt{2}}.$$

Every quantum circuit without meters has an equivalent algebraic expression. For example, if A, B, C are single-qubit gates, the circuit

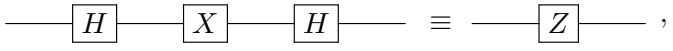


is equivalent to the algebraic expression

$$|\psi\rangle = C \cdot B \cdot A \cdot |0\rangle,$$

where $\cdot$ is the matrix product, which is usually omitted. Note that the algebraic expression equivalent to the circuit has the reverse order. Therefore, the last circuit can also be written as

where CBA is a 2×2 unitary matrix. For example, the following circuits are equivalent:



because $Z = HXH$. The equivalent algebraic expression can be used to simplify the circuit and predict its output.

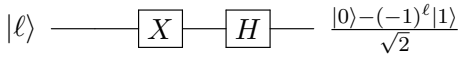
In quantum computing, we have to take advantage of classical computing notation. For instance, think that ℓ in $|\ell\rangle$ is a classical bit. Then, $|\ell\rangle$ represents both $|0\rangle$ and $|1\rangle$. This notation is used in $|\ell \oplus 1\rangle = X|\ell\rangle$. Similarly, we have $(-1)^\ell|\ell\rangle = Z|\ell\rangle$ and $(-1)^\ell \mathrm{i}\,|\ell \oplus 1\rangle = Y|\ell\rangle$. Those expressions are used to determine the output of gates X, Y, and Z when the input is a vector of the computational basis. For the gate $U(0, 0, \lambda)$, we have $\mathrm{e}^{\mathrm{i}\lambda\ell}|\ell\rangle = U(0, 0, \lambda)|\ell\rangle$, which

[4] The $T^\dagger$ gate is in fact the transpose-conjugate gate, but since T is diagonal, $T^\dagger$ is simply the conjugate gate.

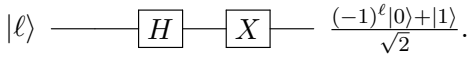
includes S, T, and their conjugate gates as subcases. All those gates don't create superposition. The only one that does is the Hadamard gate, whose output is

$$\frac{|0\rangle + (-1)^{\ell}|1\rangle}{\sqrt{2}} = H|\ell\rangle.$$

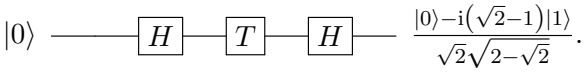
Now we are ready to show that



and



With a little more effort, we show up to a global phase that



Implementing on IBM quantum computers.

At this point, it is a good idea to use IBM's composer. After logging in IBM's website[5] (registration is needed), we have to launch the composer by clicking on Launch Composer. IBM's composer is friendly because we can drag the available gates into the circuit. Let us keep it to a basic use at this moment. Fig. 2.3 shows a circuit with the Hadamard gate followed by a measurement. We simply drag H and drop it on the first wire of the circuit, then we drag the meter and drop it after H. The meter's arrow shows that the output is re-directed to an auxiliary classical register at the bottom of the circuit.

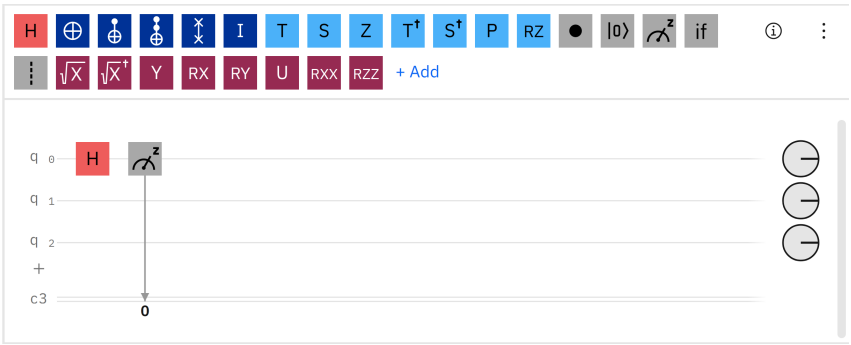


Figure 2.3: Example of a circuit with a Hadamard gate and a meter (Reprint Courtesy of IBM Corporation ©)

After the circuit is ready, we click on Setup and run, and then we have two options: (1) Run the circuit on a quantum computer by selecting one of the available quantum systems, or (2) simulate the circuit by selecting a simulator. It is usually better to start with the second option. We select *ibm_qasm_simulator* as the provider, then we select the number of shots and then we click on Run at the bottom. Fig. 2.4 shows the output of an execution. The result 000 was obtained 503 times and 001 was obtained 521 times out of 1024 shots. The first two bits must

[5] https://quantum-computing.ibm.com/

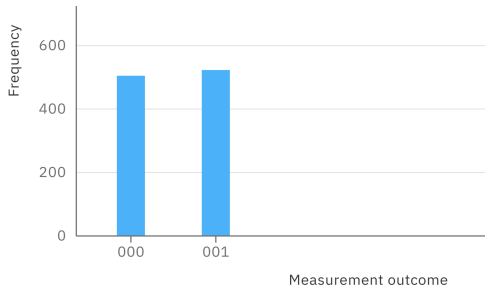


Figure 2.4: Output of the circuit with a Hadamard gate and a meter (Reprint Courtesy of IBM Corporation ©)

be discarded because they refer to qubits that have not been used (qubits q_2 and q_1). The output uses the order $q_2q_1q_0$, different from the order adopted in most textbooks on quantum computing. As we can see, the composer runs the experiment several times (up to 8192) and shows the histogram of the cumulative frequency distribution. In the case of the Hadamard gate, the most probable result is 50% each, but results close to 50% have non-negligible probabilities and frequently occur. To obtain results closer to 50%, we have to increase the number of shots.

If we choose to run on a quantum computer, the circuit will be queued and may take a long time. We can check the size of the queue when we are selecting the system. The output of the quantum computer is usually different from the simulator because errors degrade the result. Increasing the number of shots doesn't guarantee that the probability distribution tends towards the correct distribution, and it is important to check the correctness of the circuit through the simulator before running on the quantum system.

2.4 Quantum states and entanglement

The state of two qubits is described by a vector of norm 1 that belongs to a four-dimensional vector space because there are four possible results after measuring the qubits: 00, 01, 10, 11. The first bit refers to the first qubit and the second bit to the second qubit, as is usual in textbooks on quantum computing, and the least significant bit is on the right, as usual.

Following the laws of quantum mechanics, there is a one-to-one correspondence between the canonical basis and the possible measurement outcomes as follows:

$$|00\rangle = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad |01\rangle = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \quad |10\rangle = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}, \quad |11\rangle = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}.$$

In the classical case, the state of two bits is either 00 or 01 or 10 or 11, exclusively. In the quantum case, the state of two qubits is the linear combination

$$|\psi\rangle = a_0|00\rangle + a_1|01\rangle + a_2|10\rangle + a_3|11\rangle,$$

where a_0, a_1, a_2, and a_3 are complex numbers. When the state of two qubits is $|\psi\rangle$, the outcome of a measurement in the computational basis is either 00 or 01 or 10 or 11, exclusively

and stochastically; there is no way of predicting the measurement outcome even knowing $|\psi\rangle$. However, if we know $|\psi\rangle$, then we know the probabilities of outcomes, which are

$$\begin{aligned}\text{prob}(00) &= \left|\langle 00|\psi\rangle\right|^2 = |a_0|^2,\\ \text{prob}(01) &= \left|\langle 01|\psi\rangle\right|^2 = |a_1|^2,\\ \text{prob}(10) &= \left|\langle 10|\psi\rangle\right|^2 = |a_2|^2,\\ \text{prob}(11) &= \left|\langle 11|\psi\rangle\right|^2 = |a_3|^2.\end{aligned}$$

The sum of those probabilities is 1. If we don't know state $|\psi\rangle$, a single measurement doesn't allow the determination of $|\psi\rangle$, that is, we cannot find the amplitudes a_0, a_1, a_2, and a_3. There is an important theorem in quantum mechanics known as *non-cloning theorem* [20, 47, 66].

Theorem 2.1. (No cloning) Using unitary operators, it is impossible to make an identical copy of an arbitrary unknown quantum state that is available to us.

This theorem severely restricts any possibility of determining $|\psi\rangle$ through measurements. However, if we can generate $|\psi\rangle$ again and again, for example, through a circuit, we can repeat the whole process several times and obtain an approximation for prob(00), prob(01), prob(10), and prob(11). For example, by repeating 1000 times, we can determine these probabilities with two digits. Unfortunately, we are still unable to determine $|\psi\rangle$ exactly because knowing $|a_0|^2$ doesn't allow us to determine a_0 exactly. It may seem that this is unimportant—false. Let's consider a critical example. Suppose that

$$\text{prob}(00) = \frac{1}{2}, \qquad \text{prob}(01) = 0, \qquad \text{prob}(10) = 0, \qquad \text{prob}(11) = \frac{1}{2}.$$

We have at least two possibilities for $|\psi\rangle$:

$$|\psi_1\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}} \qquad \text{and} \qquad |\psi_2\rangle = \frac{|00\rangle - |11\rangle}{\sqrt{2}}.$$

Note that $|\psi_1\rangle$ and $|\psi_2\rangle$ are orthogonal. This shows that we can make a serious mistake. We don't know whether two circuits are equivalent when their probability distributions are the same.

At this point, the following question is relevant: Suppose that we know $|\psi\rangle$, is it possible to determine the state of each qubit? The answer is "depends on $|\psi\rangle$". If $|\psi\rangle$ is one of the states of the computational basis then we know the state of each qubit. For example, suppose $|\psi\rangle = |10\rangle$. We have to factorize $|\psi\rangle$ as

$$|10\rangle \;=\; |1\rangle|0\rangle \;=\; |1\rangle \otimes |0\rangle,$$

where $\otimes$ is called *Kronecker product*. When factorization is successful, we know the state of each qubit. In this case, the state of the first qubit is $|1\rangle$ and the state of the second is $|0\rangle$. When we write $|1\rangle|0\rangle$, the Kronecker product is implicitly assumed.

The Kronecker product[6] of two vectors or two matrices is defined as follows. Let A be a $m \times n$ matrix and B a $p \times q$ matrix. Then,

$$A \otimes B = \begin{bmatrix} a_{11}B & \cdots & a_{1n}B \\ & \ddots & \\ a_{m1}B & \cdots & a_{mn}B \end{bmatrix}.$$

[6]There is a more abstract formulation of the Kronecker product called *tensor product*. In this work, we use these terms interchangeably. The notation $A \otimes B$ reads "A tensor B"; however, note that the terms *tensor* and *tensor product* are used in other areas of mathematics with differing meaning, such as in differential geometry.

The result is a $mp \times nq$ matrix. The Kronecker product of vectors $|1\rangle$ and $|0\rangle$ is calculated by viewing these vectors as 2×1 matrices and is given by

$$|1\rangle \otimes |0\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \begin{bmatrix} 1 \\ 0 \end{bmatrix} \\ 1 \begin{bmatrix} 1 \\ 0 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = |10\rangle.$$

Note that the Kronecker product is noncommutative. For example, $|1\rangle \otimes |0\rangle \neq |0\rangle \otimes |1\rangle$. An important hint is to never change the order of the Kronecker product.

From the laws of quantum mechanics, if the state of a qubit is $|\psi_1\rangle$ and the state of a second one is $|\psi_2\rangle$, then the state $|\psi\rangle$ of the composite system, two interacting qubits, will initially be

$$|\psi\rangle = |\psi_1\rangle \otimes |\psi_2\rangle.$$

We can always obtain the state of the composite system when we know the states of the parts. However, the reverse process is not possible in general. For example, suppose that the state of two qubits is

$$|\psi\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}}.$$

We want to find 1-qubit states $|\psi_1\rangle = \alpha|0\rangle + \beta|1\rangle$ and $|\psi_2\rangle = \gamma|0\rangle + \delta|1\rangle$ such that

$$(\alpha|0\rangle + \beta|1\rangle) \otimes (\gamma|0\rangle + \delta|1\rangle) = \frac{|00\rangle + |11\rangle}{\sqrt{2}}.$$

Expanding the left-hand side, we obtain the following system of equations:

$$\alpha\gamma = \frac{1}{\sqrt{2}}, \quad \alpha\delta = 0, \quad \beta\gamma = 0, \quad \beta\delta = \frac{1}{\sqrt{2}}.$$

Since this system has no solution, the 2-qubit state $|\psi\rangle$ cannot be written as the Kronecker product of 1-qubit states. In quantum mechanics, a composite quantum system may have a definite state $|\psi\rangle$ while parts of the system have no definite state.

A quantum state of a composite system that cannot be factorized in terms of the Kronecker product is called an *entangled state*. Entangled states are very important in quantum computing because without them the computational power of the quantum computer would be badly impaired. However, note that the presence of entanglement in a quantum algorithm doesn't guarantee that this algorithm is more efficient than its classical counterpart.

The term "definite state" $|\psi\rangle$ in quantum mechanics means *pure state*. A state of a quantum system is called a pure state if we are 100% sure that the system is described by a norm-1 vector $|\psi\rangle$. On the other hand, if we are not 100% sure, that is, if we know that the state of the system is $|\psi_1\rangle$ with probability $0 < p < 1$ or $|\psi_2\rangle$ with probability $1 - p$, then the state is *mixed* and is represented by an *ensemble* or a matrix ρ such that $\mathrm{Tr}(\rho) = 1$. When the state of a composite quantum system is entangled, the state of any sub-system is always a mixed state.

We can generalize the discussion of this Section to n qubits, where $n \geq 1$. The computational basis has 2^n vectors, each vector with 2^n entries,

$$|0\cdots00\rangle = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \quad |0\cdots01\rangle = \begin{pmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{pmatrix}, \quad \cdots, \quad |1\cdots11\rangle = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{pmatrix}.$$

Note that the binary number inside the ket, for instance, $0\cdots0$ in $|0\cdots0\rangle$, has n bits and the state itself is the Kronecker product of n 1-qubit states. The binary number $0\cdots0$ can be written in the decimal notation as $|0\cdots0\rangle \to |0\rangle$. Each binary number inside the kets can be written in the decimal notation as

$$|0\cdots00\rangle \to |0\rangle, \quad |0\cdots01\rangle \to |1\rangle, \quad |0\cdots10\rangle \to |2\rangle, \quad ..., \quad |1\cdots11\rangle \to |2^n-1\rangle.$$

A generic state $|\psi\rangle$ belongs to a 2^n-dimensional vector space. Then,

$$|\psi\rangle = a_0\,|0\rangle + a_1\,|1\rangle + a_2\,|2\rangle + \cdots + a_{2^n-1}\,|2^n-1\rangle,$$

where

$$|a_0|^2 + |a_1|^2 + |a_2|^2 + \cdots + |a_{2^n-1}|^2 = 1.$$

After a measurement of all qubits, we obtain a random n-bit string with the following probability distribution: The outcome is either the n-bit string $0\cdots00$ with probability $|a_0|^2$, or the n-bit string $0\cdots01$ with probability $|a_1|^2$, and so on. We have a *sample space* comprising those n-bit strings and a probability distribution given by $\text{prob}(\ell) = |a_\ell|^2$, where ℓ is a n-bit string. The measurement outcome is a random variable that takes a value ℓ in this sample space with probability $\text{prob}(\ell)$.

As we have said before, a vector of the computational basis of n qubits can be written as the Kronecker product of n 1-qubit vectors. For example, for $n = 3$ qubits, we can obtain the second vector of the computational basis using the Kronecker product as

$$|0\rangle \otimes |0\rangle \otimes |1\rangle \;=\; \begin{bmatrix}1\\0\end{bmatrix} \otimes \begin{bmatrix}1\\0\end{bmatrix} \otimes \begin{bmatrix}0\\1\end{bmatrix} \;=\; \begin{bmatrix}0\\1\\0\\0\\0\\0\\0\\0\end{bmatrix} \;=\; |001\rangle.$$

In the decimal notation, $|0\rangle$ can be confused with the state of 1 qubit. To avoid confusion, we have to know what is the number of qubits. For example, if $|0\rangle$ refers to the state of 3 qubits in the decimal notation, then in binary we have $|000\rangle$.

In this Section, we have defined entangled states. It is not a good idea to try to understand this concept without using mathematics at the outset. Learning the basic definition of entanglement is similar to learning what is a prime number in basic arithmetic. Suppose we know nothing and we are going to learn the basics of arithmetic starting with addition. After attending classes, doing many exercises, and memorizing tables, we master the concept of addition. The next step is multiplication. We have to pass through the same ordeal of classes, exercises, and table memorization, but eventually, we master the concept of multiplication. Then we are ready to learn the reverse process, that is, factorization. We learn that the factors of 15 are 3 and 5. Now we try 17. We have to find two positive numbers strictly less than 17 so that when we multiply those numbers we obtain 17. We check that there are no such numbers. Then we are ready to understand the concept of prime numbers, which is very important in mathematics. If we understand prime numbers, then we are ready to understand entangled states. But this

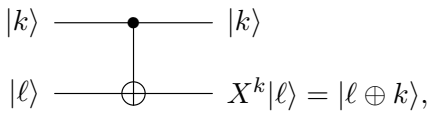
time we have to learn direct sum of vectors, and then Kronecker product of vectors, and then we try to factorize a vector. That is, we have to find two vectors $|\psi_1\rangle$ and $|\psi_2\rangle$ that have fewer entries than $|\psi\rangle$ so that $|\psi\rangle = |\psi_1\rangle \otimes |\psi_2\rangle$. The entangled states are the irreducible vectors of composite quantum systems in terms of the Kronecker product. A single-qubit state is not entangled because a single-qubit is not a composite system. We need at least two qubits. There is a long road ahead of us, of course, but two qubits is a good starting point. In the next Section, we see how to produce an entangled state in the quantum computer.

2.5 Two-qubit quantum gates

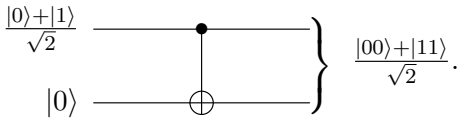
The most important 2-qubit gate is CNOT or controlled-NOT gate, also denoted by $C(X)$ or CX. It is defined as

$$\text{CNOT}\,|k\rangle|\ell\rangle = |k\rangle X^k|\ell\rangle,$$

and is represented by the circuit



where k and ℓ are bits. The state of the first qubit (*control*) doesn't change after applying CNOT. The state of the second qubit (*target*) changes only if bit k is 1. In this case the output is $X|\ell\rangle = |\ell \oplus 1\rangle$. If $k = 0$ then $X^0 = I_2$ and $I_2|\ell\rangle = |\ell\rangle$, where I_2 is the 2×2 identity matrix. We have defined CNOT by showing its action on the vectors of the computational basis. In linear algebra, this definition is complete, because to know the action of CNOT on an arbitrary vector, which is a linear combination of vectors of the computational basis, we use the linearity of this gate. For example, in the circuit below the first input is in superposition:



What is the output? The best way to determine the output is via algebraic calculations. After using the distributive property of the Kronecker product over the sum of vectors, the input to the circuit is

$$\frac{|0\rangle + |1\rangle}{\sqrt{2}} \otimes |0\rangle \;=\; \frac{1}{\sqrt{2}}\,|0\rangle \otimes |0\rangle + \frac{1}{\sqrt{2}}\,|1\rangle \otimes |0\rangle \;=\; \frac{|00\rangle + |10\rangle}{\sqrt{2}}.$$

To calculate the action of CNOT on a sum of vectors, we use the linearity of the matrix product, that is,

$$\text{CNOT} \cdot \left(\frac{|00\rangle + |10\rangle}{\sqrt{2}}\right) \;=\; \frac{1}{\sqrt{2}}\,\text{CNOT} \cdot |00\rangle + \frac{1}{\sqrt{2}}\,\text{CNOT} \cdot |10\rangle,$$

where $\text{CNOT} \cdot |00\rangle$ denotes the multiplication of the CNOT matrix by vector $|00\rangle$. Using the definition given at the beginning of the Section, we obtain $\text{CNOT}|00\rangle = |00\rangle$ and $\text{CNOT}|10\rangle = |11\rangle$, and we confirm that the output is

$$\frac{|00\rangle + |11\rangle}{\sqrt{2}}.$$

Since this result is an entangled state, we cannot factorize it and therefore we cannot write the output for each qubit.

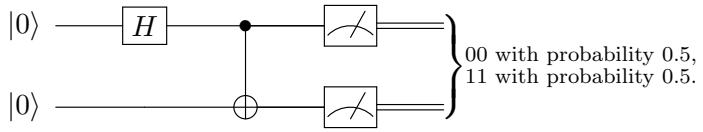
The same result is obtained if we use the matrix representation, which is

$$\text{CNOT} = \begin{bmatrix} I_2 & \\ & X \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix},$$

and the representation of $|00\rangle$ and $|10\rangle$ as 4-dimensional vectors, that is,

$$\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} + \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}.$$

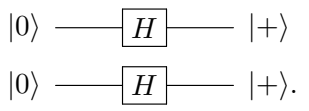
The complete circuit that implements the entangled state above when the initial state of the qubits is $|00\rangle$ is



Since the state of the first qubit is initially $|0\rangle$, we have to use H to generate $(|0\rangle + |1\rangle)/2$. In fact, we have

$$\text{CNOT} \cdot (H \otimes I)\,|00\rangle = \text{CNOT} \cdot \big(H|0\rangle \otimes |0\rangle\big) = \frac{|00\rangle + |11\rangle}{\sqrt{2}}.$$

The next example is simpler than the previous one. Consider the circuit without measurements



What is the output? Usually, to calculate the output, we convert the circuit to its equivalent algebraic expression. For this circuit, we have $(H \otimes H)|00\rangle$. The calculation is performed in the following way:

$$(H \otimes H)|00\rangle \;=\; (H \otimes H) \cdot (|0\rangle \otimes |0\rangle) \;=\; (H|0\rangle) \otimes (H|0\rangle) \;=\; |+\rangle \otimes |+\rangle.$$

In the second equality, we use the following property of the Kronecker product:

$$(A \otimes B) \cdot (C \otimes D) \;=\; (A \cdot C) \otimes (B \cdot D),$$

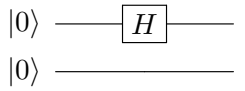
for matrices A, B, C, D, or

$$(A \otimes B) \cdot (|\psi_1\rangle \otimes |\psi_2\rangle) \;=\; (A|\psi_1\rangle) \otimes (B|\psi_2\rangle),$$

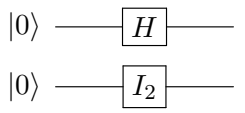
which is valid for any matrices A and B and vectors $|\psi_1\rangle$ and $|\psi_2\rangle$ as long as the number of entries of the vectors is equal to the corresponding number of columns of the matrices. So the output of the circuit is

$$|+\rangle \otimes |+\rangle \;=\; \frac{|0\rangle + |1\rangle}{\sqrt{2}} \otimes \frac{|0\rangle + |1\rangle}{\sqrt{2}} \;=\; \frac{|0\rangle + |1\rangle + |2\rangle + |3\rangle}{2}.$$

Let us consider another example. Take the following circuit without measurements:



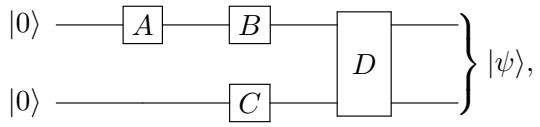
How to calculate the output using the equivalent algebraic expression? The hint is to use the following equivalent circuit:



where I_2 is the two-dimensional identity matrix. The algebraic calculation is done as follows:

$$(H \otimes I_2)|00\rangle \;=\; (H|0\rangle) \otimes (I_2|0\rangle) \;=\; |+\rangle \otimes |0\rangle \;=\; \frac{|00\rangle + |10\rangle}{\sqrt{2}}.$$

When we convert a quantum circuit to its equivalent algebraic expression, we must use the Kronecker product for gates in the same column and the matrix product for gates in the same wire or in sequence; however, we must reverse the order of the gates in the second case. For example, the algebraic expression equivalent to the circuit



where A, B, and C are 1-qubit gates and D is a 2-qubit gate, is

$$|\psi\rangle \;=\; D \cdot (B \otimes C) \cdot (A \otimes I_2) \cdot |00\rangle.$$

We can simplify this expression a little and write

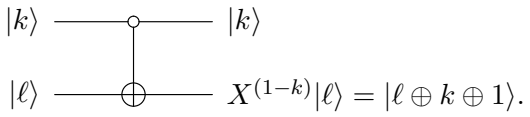
$$|\psi\rangle \;=\; D\,(BA \otimes C)|00\rangle.$$

Only D can create or destroy entanglement. Operators A, B, and C cannot create or destroy entanglement. The proof that A cannot create or destroy entanglement is as follows. Suppose that the input state is $|\psi_1\rangle \otimes |\psi_2\rangle$ (unentangled). The action of A outputs $(A|\psi_1\rangle) \otimes |\psi_2\rangle$, which is unentangled. Now suppose that the input is an entangled state $|\psi\rangle$. The action of A outputs $(A \otimes I)|\psi\rangle$. If this state is unentangled, that is, there exist $|\psi_1\rangle$ and $|\psi_2\rangle$ such that $(A \otimes I)|\psi\rangle = |\psi_1\rangle \otimes |\psi_2\rangle$, we reach a contradiction because the last equation is equivalent to $|\psi\rangle = (A^\dagger|\psi_1\rangle) \otimes |\psi_2\rangle$.

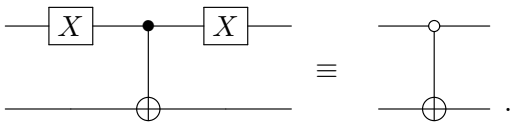
CNOT is so important that we describe a variant that is CNOT activated by 0. It is defined by

$$|k\rangle|\ell\rangle \longrightarrow |k\rangle X^{(1-k)}|\ell\rangle,$$

and is represented by the circuit



Note that the control qubit is denoted by the empty circle indicating that the CNOT's control is not activated if the state of the control qubit is $|1\rangle$. This gate is obtained from the usual CNOT by multiplying $(X \otimes I_2)$ on both sides, as shown in the following circuit equivalence:



This gate is represented by a block matrix of the form

$$(X \otimes I_2) \cdot CNOT \cdot (X \otimes I_2) = \begin{bmatrix} & I_2 \\ I_2 & \end{bmatrix} \begin{bmatrix} I_2 & \\ & X \end{bmatrix} \begin{bmatrix} & I_2 \\ I_2 & \end{bmatrix} = \begin{bmatrix} X & \\ & I_2 \end{bmatrix}.$$

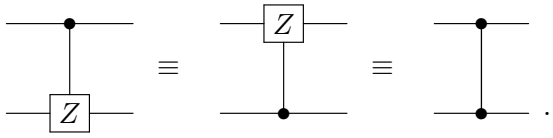
An alternative way to obtain the matrix representation is by listing sequentially the output of each vector of the computational basis

$$\begin{aligned} |00\rangle &\xrightarrow{\circ\!-\!\oplus} |01\rangle, \\ |01\rangle &\xrightarrow{\circ\!-\!\oplus} |00\rangle, \\ |10\rangle &\xrightarrow{\circ\!-\!\oplus} |10\rangle, \\ |11\rangle &\xrightarrow{\circ\!-\!\oplus} |11\rangle. \end{aligned}$$

Then convert each output into the vector notation and join them side by side to form a matrix:

$$\begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \longrightarrow \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} X & \\ & I_2 \end{bmatrix}.$$

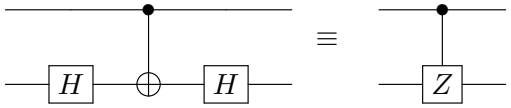
The second most important 2-qubit gate is the controlled Z gate, denoted by $C(Z)$ or CZ. Its circuit representations are



Note that it doesn't matter which qubit is the control or target, that is, Z may be controlled by the first qubit, and target on the second, or the other way around. The third representation is interesting because the qubits are on an equal footing. There is only one matrix representation given by

$$C(Z) = \begin{bmatrix} I_2 & \\ & Z \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}.$$

The CNOT and $C(Z)$ gates are connected as shown by the following circuit equivalence:



The equivalence follows from

$$(I \otimes H)\,\mathrm{CNOT}\,(I \otimes H) = C(Z),$$

which can be shown using the matrix representation. There is an alternative way of showing the equivalence by using the fact that $H^2 = I$ and then $I \otimes H$ can be replaced by $C(H)$ because two H's act trivially if the CNOT's control is not activated. Then

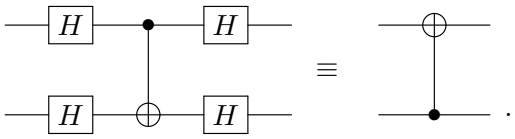
$$(I \otimes H)\,\mathrm{CNOT}\,(I \otimes H) = C(H)C(X)C(H) = C(HXH) = C(Z),$$

because $HXH = Z$ and $C(A)C(B) = C(AB)$ for any 1-qubit gates A and B.

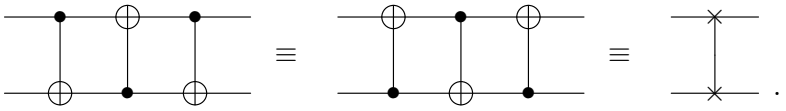
Now we are ready to show that the control and target of a CNOT invert when we multiply CNOT by $H \otimes H$ on both sides. Indeed,

$$(H \otimes H)\,\mathrm{CNOT}\,(H \otimes H) = (H \otimes I)(I \otimes H)\,\mathrm{CNOT}\,(I \otimes H)(H \otimes I) = (H \otimes I)C(Z)(H \otimes I).$$

Consider the first qubit as the target of $C(Z)$. Then we have $HZH = X$ controlled by the second qubit, and we are done. The circuit representation is



The third most important 2-qubit gate is the SWAP gate. Its circuit representations are



The matrix representation is

$$\mathrm{SWAP} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

This gate inverts an unentangled state $|\psi_1\rangle|\psi_2\rangle$ into $|\psi_2\rangle|\psi_1\rangle$, which follows from the fact that

$$\mathrm{SWAP}|00\rangle = |00\rangle, \quad \mathrm{SWAP}|01\rangle = |10\rangle, \quad \mathrm{SWAP}|10\rangle = |01\rangle, \quad \mathrm{SWAP}|11\rangle = |11\rangle.$$

Implementing on IBM quantum computers.

The implementation of the circuit with gates H and CNOT on the IBM composer is shown in Fig. 2.5. We have to drag and drop a H gate and a CNOT gate, so that the control of CNOT is qubit q_0 and the target is qubit q_1. We also have to drag and drop two meters. If we want to see the exact simulation of the probability distribution in the composer, we have to remove the meters. Note that in the composer, qubit q_0 represents the rightmost qubit, that is, the order of the qubits is $|\mathrm{q}_2\mathrm{q}_1\mathrm{q}_0\rangle$ and the output refers to $\mathrm{q}_2\mathrm{q}_1\mathrm{q}_0$. In this work and most papers and books, the opposite order is assumed. The output of a simulation with 1024 repetitions is shown in Fig. 2.6. The figure shows that the result 000 was obtained 507 times, 001 21 times, 010 51 times, and 011 445 times. We must discard the unused qubit q_2 (leftmost qubit). Therefore, the result shows that the algorithm returns 00 with probability 49,5% and 11 with probability 43,5% approximately, which is unexpected based on the analytical calculation. The wrong results 01 and 10 were obtained because quantum computers are error-prone.

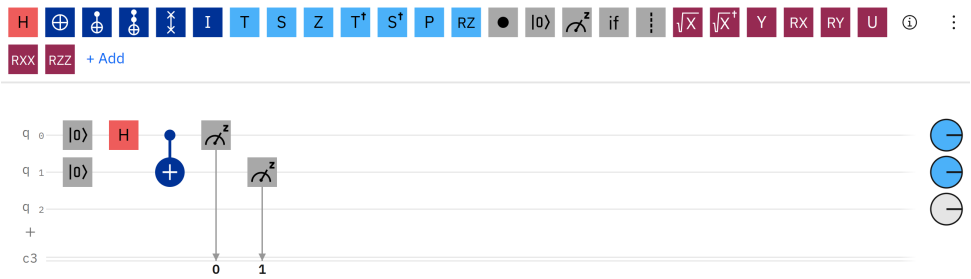


Figure 2.5: Circuit with gates H and CNOT (Reprint Courtesy of IBM Corporation ©)

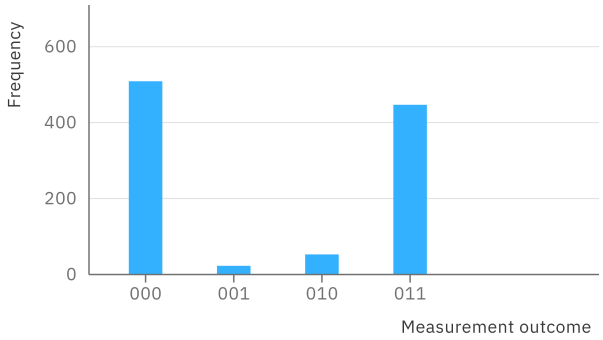


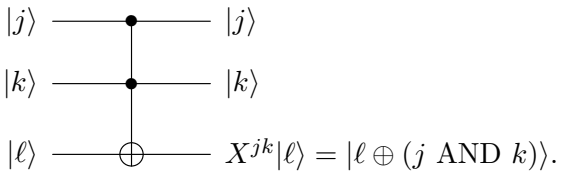
Figure 2.6: Output of the circuit of Fig. 2.5 using an IBM backend (Reprint Courtesy of IBM Corporation ©)

2.6 Multiqubit gates

The most important 3-qubit gate is the Toffoli gate, denoted by CCNOT or $C^2(X)$, defined by

$$C^2(X)\,|j\rangle|k\rangle|\ell\rangle = |j\rangle|k\rangle X^{jk}|\ell\rangle,$$

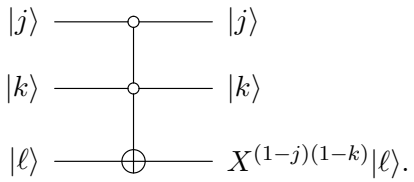
and represented by the circuit



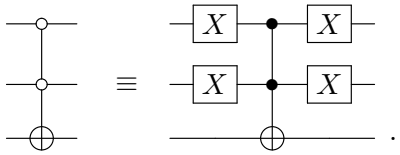
This gate has two controls and one target. X acts on the target qubit if and only if both control qubits are set to one. If one control is set to zero, the target doesn't change. This gate can be seen as the quantum version of the classical AND gate because if $\ell = 0$, the output of the third qubit is (j AND k). The matrix representation of the Toffoli gate is a diagonal block matrix given by

$$C^2(X) = \begin{bmatrix} I_2 & & & \\ & I_2 & & \\ & & I_2 & \\ & & & X \end{bmatrix}.$$

There are variants of the Toffoli gate activated when the control qubits are set to zero. For example, the circuit



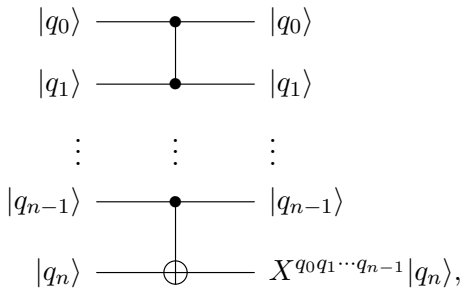
implements a gate that applies X on the third qubit if and only if the first two control qubits are set to zero. It can be implemented using a standard Toffoli gate and X gates, as shown in the following circuit equivalence:



The multiqubit Toffoli gate $C^n(X)$ is a $(n+1)$-qubit gate with n control qubits and one target. It is defined by

$$C^n(X)|q_0\rangle|q_1\rangle\cdots|q_{n-1}\rangle|q_n\rangle = |q_0\rangle|q_1\rangle\cdots|q_{n-1}\rangle X^{q_0q_1\cdots q_{n-1}}|q_n\rangle,$$

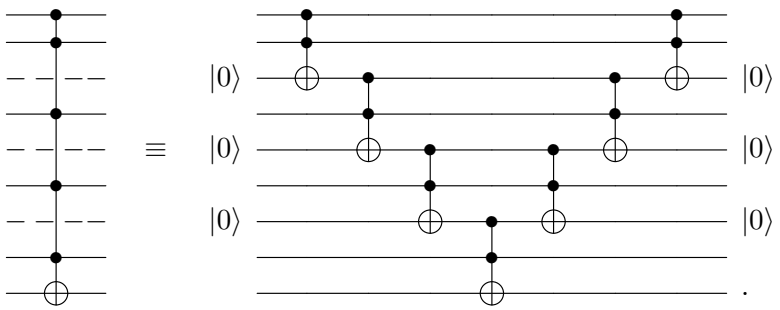
and is represented by the circuit



where $q_0q_1\cdots q_{n-1}$ is the product of bits q_0, q_1, ..., q_{n-1}. Therefore, the state of the target qubit changes if and only if all control qubits are set to 1. The state of each control qubit doesn't change when we describe this gate acting on the computational basis. The action on a generic vector is obtained by writing the vector as a linear combination of vectors of the computational basis and then using linearity.

The simplest way to decompose the multiqubit Toffoli gate in terms of the usual Toffoli gate is by using $(n-2)$ draft qubits called *ancillas*. The ancillary qubits are interlaced with the control qubits, the first ancilla qubit being inserted between the second and third qubits. The best way to explain the decomposition is to show an example. Consider the gate $C^5(X)$, whose

decomposition requires three *ancillas*, as shown in the following circuit equivalence:



The multiqubit Toffoli gate can also be activated by zero. In this case, the control qubit is shown as an empty circle. For n qubits, we have 2^n multiqubit Toffoli gates that can implement any Boolean function of n bits, as described in the next Section.

2.7 Circuit of a Boolean function

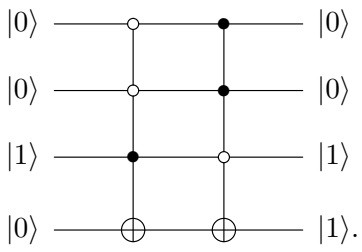
Let's show how to obtain the quantum circuit of a truth table. We only need multiqubit Toffoli gates. To show that the multiqubit Toffoli gates can implement any Boolean function on a quantum computer, let's take the 3-bit Boolean function $f(a, b, c)$ defined by the following truth table as an example:

a	b	c	$f(a,b,c)$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

After this example, it is evident how the general case is obtained. Since f has three input bits, we use multiqubit Toffoli gates with three controls. The 4th qubit is the target. The output of f is the output of a measurement of the target qubit. Since f has two clauses in the *disjunctive normal form*, we use two multiqubit Toffoli gates.[7] The first gate must be activated by the input $|001\rangle$ and the second by $|110\rangle$, which correspond to the rows of the truth table whose output is

[7] In the disjunctive normal form, we only consider rows of the truth table whose output is 1. For each row with output 1, we write a conjunction of three literals, using NOT for each input 0. Then we write a disjunction of the conjunctions, like this, $f(a, b, c) = (\bar{a} \wedge \bar{b} \wedge c) \vee (a \wedge b \wedge \bar{c})$, where $(\bar{a} \wedge \bar{b} \wedge c)$ is associated with 001 and $(a \wedge b \wedge \bar{c})$ with 110.

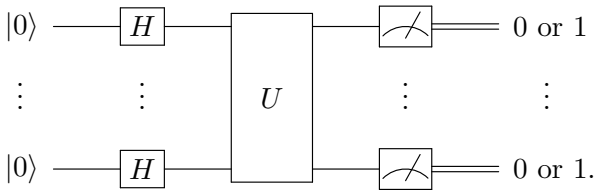
1. The following circuit implements f:



Note that if the input is $|a,b,c\rangle|0\rangle$ then the output is $|a,b,c\rangle|f(a,b,c)\rangle$. This shows that the quantum computer can calculate any n-bit Boolean function using a multiqubit Toffoli gate with $(n+1)$ qubits for each output 1 of the truth table. The goal here is not to implement classical algorithms on quantum computers because it makes no sense to build a much more expensive machine to run only classical algorithms. However, the implementation we have just described can be used for inputs in superposition, which is not allowed on a classical computer. Unfortunately, this quantum circuit construction technique for calculating truth tables is not efficient, since the number of multiqubit Toffoli gates increases exponentially as a function of the number of qubits in the worst case.

2.8 Quantum parallelism

The most standard model of quantum computing is described by the circuit



The initial state of each qubit is $|0\rangle$. Then, the Hadamard gate is applied to each qubit, preparing for the *quantum parallelism.* Then, the unitary matrix U is applied to all qubits. Finally, there is a meter for each qubit, returning a bit string.

Quantum parallelism is the simultaneous execution of an algorithm with more than one input to a single quantum processor. To execute the same task on a classical computer would require an exponential number of classical processors. To understand this concept, we focus only on U without considering the H gates. U and the meters can be interpreted as a randomized algorithm in the classical sense. If x is a n-bit string and U is a $2^n \times 2^n$ matrix, then $U|x\rangle$ is a linear combination of 2^n vectors of the computational basis. After the measurements, a n-bit string y returned, where $|y\rangle$ is one of the terms of the linear combination. That is, if the input to U is x, the output after the measurement is y. U is an "algorithm", which is executed for a single input x when the computation $U|x\rangle$ is performed. Now let us bring the H gates back.

The first step of the circuit is to perform the following computation: $(H|0\rangle) \otimes \cdots \otimes (H|0\rangle)$, that is

$$\left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right) \otimes \cdots \otimes \left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right).$$

After expanding this product, we obtain

$$\frac{1}{\sqrt{2^n}}\big(|0\cdots0\rangle + |0\cdots01\rangle + |0\cdots10\rangle + \cdots + |1\cdots1\rangle\big).$$

Each n-bit string is inside a ket of the sum above. That is, all possible inputs of a classical algorithm are represented by kets in the sum above. The next step in the standard model is to apply U. Due to the linearity of linear algebra, U is applied to all kets of the sum above *simultaneously*. Therefore, it is possible to perform 2^n simultaneous computations on a n-qubit quantum computer. Note, however, that the result of these computations is a superposition state and, after a measurement, the final output is a single n-bit string.

We can refine the standard model by adding an extra register for draft calculations. Since unitary gates are invertible, the whole calculation process without measurement is reversible. This means that no information is erased. The number of output qubits must be equal to the number of input qubits. We postpone the discussion about this refinement and present it after describing and analyzing the basic quantum algorithms.

Chapter 3

Deutsch's Algorithm

Deutsch's algorithm is the first algorithm exploiting quantum parallelism. It uses two qubits (only one in the economical version) and has a very modest gain, but it has inspired the construction of several new quantum algorithms that are more efficient than their classical counterparts. Deutsch's problem was posed in 1985 [17] without yet using the quantum circuit model. The concepts of universal gates and quantum circuits were first presented in [18], approximately four years later. A generalization of Deutsch's algorithm was described in [19] and is known as the Deutsch-Jozsa algorithm (see next Chapter). The version described here follows the modern view of Deutsch's algorithm [14, 45], which differs slightly from the original one. In the final Section, we describe an implementation of Deutsch's algorithm with only one qubit.

3.1 Problem formulation

Suppose we have a 1-bit Boolean function $f : \{0,1\} \longrightarrow \{0,1\}$ without knowing the details of the implementation of f. We want to determine whether this function is *balanced* or *constant*. A 1-bit Boolean function is balanced if $f(0) \neq f(1)$; otherwise, the function is constant, in this case $f(0) = f(1)$. There are four Boolean functions f, whose truth tables are described in Fig. 3.1 with names f_0 to f_3.

x	$f_0(x)$	$f_1(x)$	$f_2(x)$	$f_3(x)$
0	0	0	1	1
1	0	1	0	1

Figure 3.1: Truth tables of all 1-bit Boolean functions.

The *disjunctive normal forms* are

$$\begin{aligned} f_0(x) &= 0, \\ f_1(x) &= x, \\ f_2(x) &= \bar{x}, \\ f_3(x) &= \bar{x} \vee x, \end{aligned}$$

where $\bar{x} = \text{NOT}\, x$. The Boolean expression of f_3 can be simplified since $\bar{x} \vee x = 1$.

A classical algorithm that finds the solution to this problem needs to evaluate f twice, that is, it is really necessary to evaluate $f(0)$ and $f(1)$. However, Deutsch's algorithm uses a unitary

operator U_f that implements f and calls this operator only once. In this case, $f(0)$ and $f(1)$ are also evaluated, but the difference is that the evaluations are performed simultaneously. This idea is used recurrently in quantum algorithms.

In the quantum case, f is implemented through a 2-qubit unitary operator U_f defined as

$$U_f|x\rangle|j\rangle = |x\rangle|j \oplus f(x)\rangle,$$

where $\oplus$ is the XOR operation or addition modulo 2. This is a recipe that can be used to implement an arbitrary n-bit Boolean function. We have to take care that x is the input to the first register, and to obtain $f(x)$ we set $j = 0$ as the input to the second register and then we look at the output of the second register. Now we use the technique described in Chapter 2 to obtain the quantum circuit of functions f_0 to f_3. We use their disjunctive normal forms and we have to use one multiqubit Toffoli gate for each output 1 in the truth table. In the 2-qubit case, a multiqubit Toffoli gate is a CNOT activated by either 0 or 1. There is no output 1 in the truth table of f_0. Then,

$$U_{f_0} = I \otimes I.$$

There is one output 1 in the truth table of f_1, which has input 1. We use the standard CNOT, which is activated when the control is set to 1. Then,

$$U_{f_1} = \text{CNOT}.$$

There is one output 1 in the truth table of f_2, which has input 0. We use the CNOT that is activated when the control is set to 0. Then,

$$U_{f_2} = (X \otimes I) \cdot \text{CNOT} \cdot (X \otimes I).$$

Finally, there are two outputs 1 in the truth table of f_3 with inputs 0 and 1. We use the standard CNOT and the CNOT that is activated when the control is set to 0. Then,

$$U_{f_3} = (X \otimes I) \cdot \text{CNOT} \cdot (X \otimes I) \cdot \text{CNOT}.$$

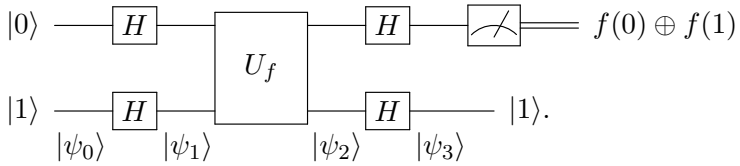
It is clear that the general recipe to implement Boolean functions using their truth tables produces unnecessary large circuits. In this case, we need to simplify the Boolean expression before building the circuit. There is no recipe at this point to guide us except for our ability in handling Boolean functions and making circuits. For f_3, we know that $f_3(x) = 1$ and the output must be 1 regardless of the input x to the first qubit. Since the input to the second qubit is 0, output 1 is obtained by using a X. Then, the simplified version of U_{f_3} is

$$U_{f_3} = I \otimes X.$$

Note that U_f is unitary in all cases.

3.2 The algorithm

Deutsch's algorithm is described in Algorithm 3.1 and the (non-economical) circuit is



Algorithm 3.1: Deutsch's algorithm

Input: A Boolean function $f : \{0,1\} \longrightarrow \{0,1\}$.
Output: 0 if f is constant, 1 if f is balanced.

1 Prepare the initial state $|0\rangle|1\rangle$;
2 Apply $H \otimes H$;
3 Apply U_f;
4 Apply $H \otimes H$;
5 Measure the first qubit in the computational basis.

We easily check that the circuit corresponds exactly to the steps of Algorithm 3.1. The output $f(0) \oplus f(1)$ is 0 if f is constant, and 1 if f is balanced. Note that the last Hadamard gate applied to the second qubit can be eliminated without affecting the algorithm. This gate is here because the central part of the circuit is symmetric and the analysis of the algorithm is neater with it than without it. The states at the bottom of the circuit are used in the analysis of the algorithm.

3.3 Analysis of the algorithm

After the first step, the state of the qubits is

$$|\psi_0\rangle = |0\rangle \otimes |1\rangle.$$

After the second step, the state of the qubits is

$$\begin{aligned} |\psi_1\rangle &= (H|0\rangle) \otimes (H|1\rangle) \\ &= \frac{|0\rangle + |1\rangle}{\sqrt{2}} \otimes |-\rangle, \end{aligned}$$

where $|-\rangle = (|0\rangle - |1\rangle)/\sqrt{2}$. After the third step, the state of the qubits is

$$\begin{aligned} |\psi_2\rangle &= U_f|\psi_1\rangle \\ &= \frac{U_f|0\rangle|-\rangle + U_f|1\rangle|-\rangle}{\sqrt{2}}. \end{aligned}$$

To simplify $|\psi_2\rangle$, let us show the following proposition:

Proposition 3.1. Let $f : \{0,1\}^n \longrightarrow \{0,1\}$ be a n-bit Boolean function and define U_f as

$$U_f = \sum_{x \in \{0,1\}^n} \sum_{j=0}^{1} |x, j \oplus f(x)\rangle\langle x, j|.$$

Then

$$U_f\big(|x\rangle \otimes |-\rangle\big) = (-1)^{f(x)}|x\rangle \otimes |-\rangle.$$

Proof. Using the definition of $|-\rangle$, we obtain

$$U_f\big(|x\rangle \otimes |-\rangle\big) \;=\; \frac{U_f|x\rangle|0\rangle - U_f|x\rangle|1\rangle}{\sqrt{2}}.$$

Using the definition of U_f, we obtain

$$U_f\big(|x\rangle \otimes |-\rangle\big) \;=\; \frac{|x\rangle|f(x)\rangle - |x\rangle|1 \oplus f(x)\rangle}{\sqrt{2}}.$$

If $f(x) = 0$, the right-hand side is $|x\rangle|-\rangle$ and if $f(x) = 1$, the right-hand side is $-|x\rangle|-\rangle$. Then, joining these results together, we have $(-1)^{f(x)}|x\rangle \otimes |-\rangle$. □

Using the proposition above, we simplify $|\psi_2\rangle$ to

$$\begin{aligned} |\psi_2\rangle &= \frac{(-1)^{f(0)}|0\rangle + (-1)^{f(1)}|1\rangle}{\sqrt{2}} \otimes |-\rangle \\ &= \begin{cases} \pm\,|+\rangle \otimes |-\rangle, & \text{if } f(0) = f(1), \\ \pm\,|-\rangle \otimes |-\rangle, & \text{if } f(0) \neq f(1). \end{cases} \end{aligned}$$

In the last passage, we have simplified the state of the first qubit without specifying exactly what is the sign of the amplitude. This sign has no effect on the output of the algorithm. After the fourth step, the state of the qubits is

$$\begin{aligned} |\psi_3\rangle &= (H \otimes H)|\psi_2\rangle \\ &= \begin{cases} \pm\,|0\rangle \otimes |1\rangle, & \text{if } f(0) = f(1), \\ \pm\,|1\rangle \otimes |1\rangle, & \text{if } f(0) \neq f(1), \end{cases} \end{aligned}$$

where we have used the fact that $H|+\rangle = |0\rangle$ and $H|-\rangle = |1\rangle$, which can be deducted using $|+\rangle = H|0\rangle$, $|-\rangle = H|1\rangle$, and $H^2 = I$. The state $|\psi_3\rangle$ can be further simplified to

$$|\psi_3\rangle \;=\; \pm\,|f(0) \oplus f(1)\rangle \otimes |1\rangle.$$

After the fifth step, the measurement of the first qubit in the computational basis returns $f(0) \oplus f(1)$, which is 0 if f is constant and 1 if f is balanced, concluding the analysis of the algorithm.

Note that the sign of $|\psi_3\rangle$ has no influence on the measurement result because the probability of obtaining $f(0) \oplus f(1)$ is $|\pm 1|^2 = 1$. It is easy to check, if relevant, that this sign is $(-1)^{f(0)}$.

3.4 Analysis of the entanglement

Summarizing the set of states of the qubits after each step, we have

$$\begin{aligned} &|\psi_0\rangle = |0\rangle \otimes |1\rangle, \\ &|\psi_1\rangle = |+\rangle \otimes |-\rangle, \\ &|\psi_2\rangle = \begin{cases} \pm\,|+\rangle \otimes |-\rangle, & \text{if } f(0) = f(1), \\ \pm\,|-\rangle \otimes |-\rangle, & \text{if } f(0) \neq f(1), \end{cases} \\ &|\psi_3\rangle = \pm\,|f(0) \oplus f(1)\rangle \otimes |1\rangle. \end{aligned}$$

Regardless of f, the qubits are unentangled during the execution of Deutsch's algorithm because each state $|\psi_i\rangle$ for i from 1 to 3 is a tensor product of 1-qubit pure states. The qubits are unentangled all the time. This means that Deutsch's algorithm is faster than classical algorithms making use of quantum parallelism only.

There is an alternate route to analyze entanglement. No 2-qubit operator $A \otimes B$ creates or destroys entanglement. In Deutsch's algorithm, only CNOT can create entanglement and it is used at most one time. Then, for each function f we can simplify the whole algorithm in order to obtain only one final operator, whose input is $|01\rangle$. If we simplify the algorithm for each f, we obtain

$$\begin{aligned}
(H \otimes H)\, U_{f_0} (H \otimes H) &= (H \otimes H) \cdot (H \otimes H) = I \otimes I, \\
(H \otimes H)\, U_{f_1} (H \otimes H) &= (H \otimes H) \cdot \mathrm{CNOT} \cdot (H \otimes H) = \mathrm{CNOT}_{10}, \\
(H \otimes H)\, U_{f_2} (H \otimes H) &= (Z \otimes I) \cdot \mathrm{CNOT}_{10} \cdot (Z \otimes I), \\
(H \otimes H)\, U_{f_3} (H \otimes H) &= I \otimes Z,
\end{aligned}$$

where the control of CNOT_{10} is the second qubit and the target is the first qubit. We conclude right away that no entanglement is created when f is f_0 or f_3. For f_1 and f_2, the CNOT's control qubit is set to 1 because the input is $|01\rangle$, in these cases, no entanglement is created because CNOT creates entanglement only if the state of the control qubit is a superposition state.

We can take advantage of the simplified version of the algorithm to check the output again and reanalyze the algorithm. The states of the qubits just before the measurement are

$$\begin{aligned}
|\psi_3\rangle\big|_{f_0} &= (I \otimes I)|0,1\rangle = |0,1\rangle, \\
|\psi_3\rangle\big|_{f_1} &= \mathrm{CNOT}_{10}|0,1\rangle = |1,1\rangle, \\
|\psi_3\rangle\big|_{f_2} &= (Z \otimes I) \cdot \mathrm{CNOT}_{10} \cdot (Z \otimes I)|0,1\rangle = -|1,1\rangle, \\
|\psi_3\rangle\big|_{f_3} &= (I \otimes Z)|0,1\rangle = -|0,1\rangle.
\end{aligned}$$

The output of a measurement of the first qubit is 0 for f_0 and f_3, which are the constant functions, and 1 for f_1 and f_2, which are the balanced functions. We can also check that the sign is $(-1)^{f(0)}$ because $f_0(0) = f_1(0) = +1$ and $f_2(0) = f_3(0) = -1$.

3.5 Who implements the oracle?

Deutsch's algorithm is said to be more efficient than its classical counterpart in a limited context called "query complexity" [30]. In this context, we must compare the number of evaluations of f in the classical case with the number of times U_f is used in the quantum case. This is a rule of the game. Deutsch's algorithm applies U_f only once and the classical algorithm needs to evaluate f two times. Then, Deutsch's is faster.

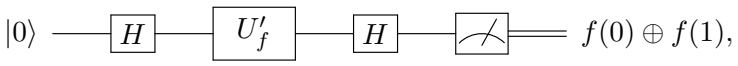
Note that the analysis of Deutsch's algorithm shows that f is evaluated at two distinct points in the domain simultaneously. This is not possible to be carried out using a sequential classical algorithm. However, it is possible to be carried out in a classical computer with parallel processors if the number of simultaneous threads does not scale up. Since Deutsch's algorithm uses a fixed number of qubits, we cannot scale it up and it is not possible to perform an asymptotic analysis. In terms of time complexity, the quantum version has no gain. The importance of

Deutsch's algorithm lies in the fact that it stimulated the search for generalizations, such as the Deutsch-Jozsa, Bernstein-Vazirani, and Simon algorithms, which inspired Shor in the elaboration of a quantum algorithm for factoring composite integers and a quantum algorithm for calculating discrete logarithms.

Last but not least, we remark that it is not up to us to implement the oracle. It is someone else's job. Without this understanding, we face a contradiction—we have to know the answer before even starting to implement the algorithm that will find the answer. It is allowed to query function f implemented by someone without looking at its implementation details. We are given what is called a *black box* quantum computer with U_f already implemented, which we can use more than once, and we may add new gates, but cannot look at the innards of the black box that implements U_f.

3.6 Economical circuit of Deutsch's algorithm

Deutsch's algorithm can be implemented with only one qubit in the following way:



where

$$U'_f = \sum_{x=0}^{1} (-1)^{f(x)} |x\rangle\langle x|.$$

From Proposition 3.1, we see that the second qubit is not necessary for the algorithm, although it is necessary if we want to obtain $f(x)$ in the classical sense, as shown ahead. By expanding the sum, we obtain

$$U'_f = \begin{bmatrix} (-1)^{f(0)} & 0 \\ 0 & (-1)^{f(1)} \end{bmatrix}.$$

Then,

$$U'_f = \begin{cases} \pm I, \text{ if } f(0) = f(1), \\ \pm Z, \text{ if } f(0) \neq f(1). \end{cases}$$

The analysis of the algorithm reduces to calculating

$$HU'_f H|0\rangle = \begin{cases} \pm\, I|0\rangle, & \text{if } f(0) = f(1), \\ \pm\, X|0\rangle, & \text{if } f(0) \neq f(1). \end{cases}$$

Using that $f(0) \oplus f(1) = 0$ if $f(0) = f(1)$, and $f(0) \oplus f(1) = 1$ if $f(0) \neq f(1)$, we obtain

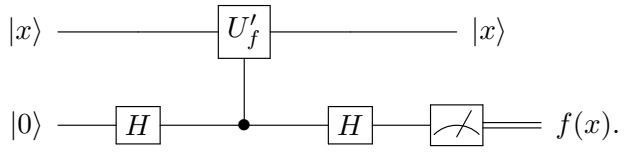
$$HU'_f H|0\rangle = \pm|f(0) \oplus f(1)\rangle.$$

After a measurement, the output is $f(0) \oplus f(1)$ with probability $|\pm 1|^2 = 1$.

Now it is straightforward to check that there is no entanglement in Deutsch's algorithm because a qubit cannot entangle with itself. Entanglement requires at least two qubits.

Consulting the oracle

In the non-economical circuit, if the input to the first qubit of U_f is $|x\rangle$, U_f returns $f(x)$ when we perform a measurement of the second qubit. In the economical circuit, we obtain $f(x)$ in the classical sense using the circuit



Indeed, the steps of this circuit are

$$|x\rangle|0\rangle \xrightarrow{I\otimes H} |x\rangle|+\rangle \xrightarrow{U'_f} \frac{|x\rangle|0\rangle + (-1)^{f(x)}|x\rangle|1\rangle}{\sqrt{2}} \xrightarrow{I\otimes H} |x\rangle|f(x)\rangle.$$

To calculate the last step, we use that $f(x)$ is either 0 or 1 for a fixed x. Firstly, we suppose that $f(x) = 0$, then $I \otimes H$ is applied to $|x\rangle|+\rangle$ resulting in $|x\rangle|f(x)\rangle$, and secondly, we suppose that $f(x) = 1$, then $I \otimes H$ is applied to $|x\rangle|-\rangle$ resulting in $|x\rangle|f(x)\rangle$. After measuring the second qubit in the computational basis, we obtain $f(x)$ with probability 1.

Chapter 4

Deutsch-Jozsa Algorithm

The Deutsch-Jozsa algorithm is a deterministic quantum algorithm, a generalization of Deutsch's algorithm, and the first example that is exponentially faster than its equivalent classical deterministic algorithm. It was published in 1992 [19] and revisited in 1998 [14]. Many books [30, 34, 37, 40, 45, 65] and papers [43, 52, 53] have reviewed and generalized this algorithm.

4.1 Problem formulation

Let $f : \{0,1\}^n \longrightarrow \{0,1\}$ be a n-bit Boolean function, $n \geq 2$, with the following property: f is either balanced or constant. A Boolean function is balanced if the inverse image of point 0 has cardinality 2^{n-1}, and it is constant if the inverse image of point 0 has cardinality 0 or 2^n. Suppose we are able to evaluate this function at any point in the domain; however, we don't have access to the details of the implementation of f, that is, we are given a *black-box* quantum computer with f already implemented. The Deutsch-Jozsa algorithm solves the following problem: Determine whether f is balanced or constant using this black-box quantum computer.

There are only two constant functions, which are $f(x) = 0$ and $f(x) = 1$ for all n-bit string x, but there are many balanced functions. The best classical deterministic algorithm that solves this problem, given a black-box function f that is either balanced or constant implemented on a n-bit classical computer, is the following: Evaluate f at $2^{n-1}+1$ distinct points in its domain and check whether the output is always the same (f is constant) or not (f is balanced).

The Deutsch-Jozsa algorithm, on the other hand, is a deterministic quantum algorithm that uses a black-box unitary operator U_f only once. The action of U_f on the computational basis is

$$U_f|x\rangle|j\rangle = |x\rangle|j \oplus f(x)\rangle,$$

where $x \in \{0,1\}^n$ and $j \in \{0,1\}$. The qubits are split into two quantum registers with sizes n and 1.[1] After making a superposition of vectors $|x\rangle$ for all x, the Deutsch-Jozsa algorithm is able to compute $f(x)$ for all x with a single application of U_f and after quick post-processing is able to determine whether f is constant or balanced.

In the next proposition, we show that U_f is unitary. Then, since each entry of U_f is either 0 or 1, U_f is a 2^{n+1}-dimensional permutation matrix.[2]

[1] A quantum register is a set of qubits.

[2] A *permutation matrix* is a square binary matrix such that each row and each column has exactly one entry equal to 1 and zeroes elsewhere.

Proposition 4.1. U_f is unitary for any n-bit Boolean function.

Proof. Let us show that $U_f^\dagger U_f = I$. Using the definition of U_f, we have

$$\begin{aligned}\langle x'|\langle j'|U_f^\dagger U_f|x\rangle|j\rangle &= \langle x'|x\rangle\ \langle j'\oplus f(x')|j\oplus f(x)\rangle \\ &= \delta_{xx'}\ \langle j'\oplus f(x')|j\oplus f(x)\rangle\,.\end{aligned}$$

Using that $\delta_{xx'}\neq 0$ only if $x=x'$, we have $\delta_{xx'}\langle j'\oplus f(x')|j\oplus f(x)\rangle = \delta_{xx'}\langle j'|j\rangle$. Then

$$\langle x'|\langle j'|U_f^\dagger U_f|x\rangle|j\rangle = \delta_{xx'}\delta_{jj'}.$$

Since j, j' are arbitrary bits and x, x' arbitrary n-bit strings, the proof is done. □

4.2 The quantum algorithm

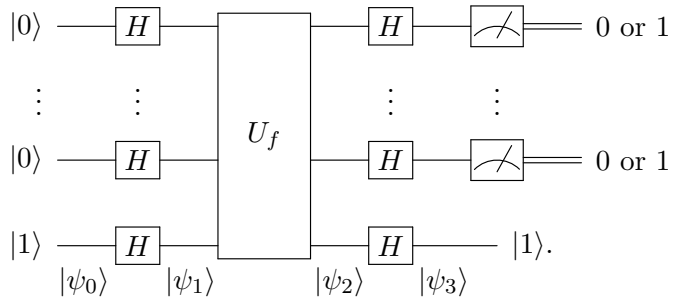
Algorithm 4.1: Deutsch-Jozsa algorithm

Input: A black-box U_f implementing a n-bit Boolean function $f:\{0,1\}^n \longrightarrow \{0,1\}$, which is either balanced or constant.

Output: 0 if f is constant; otherwise, f is balanced.

1 Prepare the initial state $|0\rangle^{\otimes n}|1\rangle$;
2 Apply $H^{\otimes(n+1)}$;
3 Apply U_f;
4 Apply $H^{\otimes(n+1)}$;
5 Measure the first register in the computational basis.

The Deutsch-Jozsa algorithm is described in Algorithm 4.1 and the $(n+1)$-qubit circuit is



We easily check that the circuit corresponds exactly to the steps of Algorithm 4.1. The output is the n-bit string 0 if f is constant, and different from 0 if f is balanced. Note that the last Hadamard gate applied to the last qubit can be eliminated without affecting the algorithm. This gate is here because the central part of the circuit is symmetric and the analysis of the algorithm is neater with it than without it. The states at the bottom of the circuit are used in the analysis of the algorithm.

4.3 Analysis of the algorithm

After the first step, the state of the qubits is

$$|\psi_0\rangle = |0\rangle^{\otimes n} \otimes |1\rangle.$$

After the second step, the state of the qubits is

$$\begin{aligned} |\psi_1\rangle &= (H|0\rangle)^{\otimes n} \otimes (H|1\rangle) \\ &= \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle|-\rangle, \end{aligned}$$

where $|-\rangle = (|0\rangle - |1\rangle)/\sqrt{2}$. After the third step, the state of the qubits is

$$\begin{aligned} |\psi_2\rangle &= U_f|\psi_1\rangle \\ &= \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} U_f\big(|x\rangle|-\rangle\big). \end{aligned}$$

To simplify $|\psi_2\rangle$, we use Proposition 3.1 on Page 28, which states that

$$U_f\big(|x\rangle|-\rangle\big) = (-1)^{f(x)}|x\rangle|-\rangle.$$

We obtain

$$|\psi_2\rangle = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} (-1)^{f(x)}|x\rangle|-\rangle.$$

After the fourth step, the state of the qubits is

$$\begin{aligned} |\psi_3\rangle &= H^{\otimes(n+1)}|\psi_2\rangle \\ &= \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} (-1)^{f(x)}\big(H^{\otimes n}|x\rangle\big) \otimes \big(H|-\rangle\big). \end{aligned}$$

To simplify $|\psi_3\rangle$, let us show the following proposition:

Proposition 4.2. Let $x \in \{0,1\}^n$ be a n-bit string $x_0 \cdots x_{n-1}$. Then

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

where $x \cdot y = x_0y_0 + \cdots + x_{n-1}y_{n-1} \mod 2$.

Proof. Using that $x = (x_0 \cdots x_{n-1})_2$, we obtain

$$\begin{aligned} H^{\otimes n}|x\rangle &= \big(H|x_0\rangle\big) \otimes \cdots \otimes \big(H|x_{n-1}\rangle\big) \\ &= \Big(\frac{1}{\sqrt{2}} \sum_{y_0=0}^{1} (-1)^{x_0y_0}|y_0\rangle\Big) \otimes \cdots \otimes \Big(\frac{1}{\sqrt{2}} \sum_{y_{n-1}=0}^{1} (-1)^{x_{n-1}y_{n-1}}|y_{n-1}\rangle\Big). \end{aligned}$$

Putting all sums at the beginning, we obtain

$$H^{\otimes n}|x\rangle \;=\; \frac{1}{\sqrt{2^n}} \sum_{y_0=0}^{1} \cdots \sum_{y_{n-1}=0}^{1} (-1)^{(x_0y_0+\cdots+x_{n-1}y_{n-1})} |y_0\rangle \otimes \cdots \otimes |y_{n-1}\rangle.$$

Using the definition of $x \cdot y$ and converting to the decimal notation, we complete the proof. □

Using the proposition above, we simplify $|\psi_3\rangle$ to

$$|\psi_3\rangle \;=\; \frac{1}{2^n} \sum_{y=0}^{2^n-1} \left(\sum_{x=0}^{2^n-1} (-1)^{x\cdot y+f(x)} \right) |y\rangle \otimes |1\rangle.$$

The amplitude of state $|0\rangle|1\rangle$ (0 in decimal) is

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

The probability that a measurement of the first register returns $y = 0$ (in decimal) is

$$p(0) \;=\; \left| \frac{1}{2^n} \sum_{x=0}^{2^n-1} (-1)^{f(x)} \right|^2.$$

If f is constant, $p(0) = 1$ and we know with certainty that the output is $y = 0$. If f is balanced, $p(0) = 0$ and we know with certainty that the output is $y \neq 0$.

4.4 Analysis of the entanglement

There is no entanglement between the registers because $|\psi_0\rangle$ to $|\psi_3\rangle$ can be written either as $|\psi\rangle \otimes |1\rangle$ or $|\psi\rangle \otimes |-\rangle$, for some state $|\psi\rangle$. We have to check the first register only. From the circuit of the algorithm, we realize that the only operator that creates or destroys entanglement is U_f. Then, it is enough to analyze

$$|\psi\rangle \;=\; \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} (-1)^{f(x)} |x\rangle,$$

which is the state of the first register after applying U_f. We ask ourselves whether there are a_i and b_i so that

$$|\psi\rangle \;=\; \frac{a_0|0\rangle + b_0|1\rangle}{\sqrt{2}} \otimes \cdots \otimes \frac{a_{n-1}|0\rangle + b_{n-1}|1\rangle}{\sqrt{2}},$$

where each pair (a_i, b_i) must obey $|a_i|^2+|b_i|^2 = 2$. This is the only way of having no entanglement at all. Equivalently, we ask whether the system of equations

$$\begin{aligned} a_0...a_{n-2}a_{n-1} &= (-1)^{f(0...00)} \\ a_0...a_{n-2}b_{n-1} &= (-1)^{f(0...01)} \\ a_0...b_{n-2}a_{n-1} &= (-1)^{f(0...10)} \\ a_0...b_{n-2}b_{n-1} &= (-1)^{f(0...11)} \\ &\vdots \\ b_0...b_{n-2}b_{n-1} &= (-1)^{f(1...11)} \end{aligned}$$

admits a solution or not. It is straightforward to check that $a_i \neq 0$ and $b_i \neq 0$ for all i. Besides, we need not worry about phases of a_i. In fact, without loss of generality, we consider a_i real and positive because a global factor can be discarded. Then, by selecting equation

$$b_0a_1a_2...a_{n-2}a_{n-1} = (-1)^{f(1...00)},$$

we conclude that b_0 is also real. The same applies to the other b_i. Now, let us show that $a_i = 1$ and $b_i = \pm 1$. Let us start with a_0 and b_0 by selecting the following equations:

$$\begin{aligned} a_0a_1a_2...a_{n-2}a_{n-1} &= (-1)^{f(0...00)}, \\ b_0a_1a_2...a_{n-2}a_{n-1} &= (-1)^{f(1...00)}. \end{aligned}$$

Dividing them, we obtain $a_0 \pm b_0 = 0$. This result together with the constraint $a_0^2 + b_0^2 = 2$ implies that $a_0 = 1$ and $b_0 = \pm 1$. The same applies to the other a_i and b_i. Now, counting the number of unentangled states, we obtain at most 2^n, up to a global sign. Or, at most 2^{n+1}.

There are only two constant functions: $f(x) = 0$ and $f(x) = 1$ for all x, which correspond to $|\psi\rangle = \big(H|0\rangle\big)^{\otimes n}$ and $|\psi\rangle = -\big(H|0\rangle\big)^{\otimes n}$, respectively. In both cases, there is no entanglement. Then, let us count the number of balanced functions. The exact number is the number of subsets with cardinality 2^{n-1} because as soon as we find such a subset S, we define a balanced function taking $f(x) = 0$ if $x \in S$ and $f(x) = 1$ otherwise. When we cover all such subsets, we have obtained all balanced functions. Since the domain has cardinality 2^n, the number of balanced functions is $\binom{2^n}{2^{n-1}}$.[3]

The number of balanced functions grows quicker than the number of unentangled states. Indeed, using the asymptotic approximation[4]

$$\binom{2p}{p} \approx \frac{2^{2p}}{\sqrt{\pi p}}$$

we obtain

$$\binom{2^n}{2^{n-1}} \approx \frac{\sqrt{2}}{\sqrt{\pi}} \frac{2^{2^n}}{\sqrt{2^n}}.$$

We conclude that when n increases, there are more and more balanced functions f with U_f creating entanglement.

[3] An alternative way of counting the number of balanced functions is by considering the number of permutations of a list with 2^{n-1} zeros and 2^{n-1} ones.

[4] https://en.wikipedia.org/wiki/Binomial_coefficient

The discussion above shows that there is $n_0 \geq 2$, such that for all $n \geq n_0$, U_f creates entanglement. Since, $\binom{2^n}{2^{n-1}} > 2^{n+1}$ for $n \geq 3$, we can take $n_0 = 3$. The only remaining case that may have no entanglement is $n = 2$. For $n = 2$, there are 6 balanced functions, whose truth tables are described in Fig. 4.4 with function names f_0 to f_5.

x_0 x_1	$f_0(x)$	$f_1(x)$	$f_2(x)$	$f_3(x)$	$f_4(x)$	$f_5(x)$
0 0	0	0	0	1	1	1
0 1	0	1	1	0	0	1
1 0	1	0	1	0	1	0
1 1	1	1	0	1	0	0

Figure 4.1: Truth tables of all 2-bit balanced functions.

It is enough to analyze f_0 to f_2 because the other functions complement those ones. For instance, f_5 is the complement of f_0, which means that state $|\psi\rangle$ created by U_{f_5} is equal to the state created by U_{f_0} up to a global phase. States $|\psi\rangle$ that correspond to f_0 to f_2 (without normalization) are

$$|00\rangle + |01\rangle - |10\rangle - |11\rangle = \big(|0\rangle - |1\rangle\big) \otimes \big(|0\rangle + |1\rangle\big),$$
$$|00\rangle - |01\rangle + |10\rangle - |11\rangle = \big(|0\rangle + |1\rangle\big) \otimes \big(|0\rangle - |1\rangle\big),$$
$$|00\rangle - |01\rangle - |10\rangle + |11\rangle = \big(|0\rangle - |1\rangle\big) \otimes \big(|0\rangle - |1\rangle\big),$$

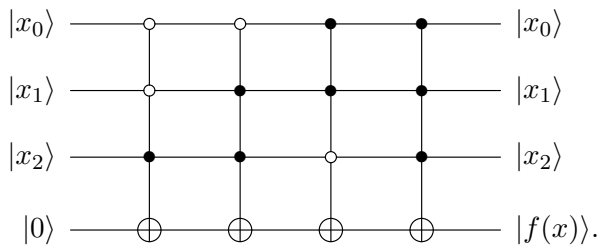
respectively. All those states are unentangled. We conclude that there is no entanglement in the Deutsch-Jozsa algorithm when $n = 2$.

4.5 Implementing the oracle

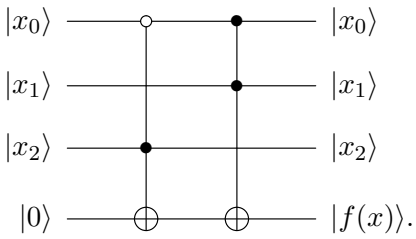
We repeat here what has already been written before about the oracle. It is not up to us to implement the oracle. It is someone else's job. Without this understanding, we face a contradiction—we have to know the answer before even starting to implement the algorithm that will find the answer. It is allowed to us to evaluate function f implemented by someone without looking at its implementation details. We are given what is called a *black-box* quantum computer with U_f already implemented, which we can use and add new gates, but cannot see inside. In the classical case, we have to count the number of evaluations of f. In the quantum case, we have to count the number of applications of U_f. This is how we calculate the query complexity of oracle-based algorithms. Note that it does not matter whether the evaluation of f is efficient or not.

With this understanding, let us show how to implement oracle U_f for the following entangling balanced function: $f(x) = 0$ if $x \in \{000, 010, 100, 101\}$ and $f(x) = 1$ if $x \in \{001, 011, 110, 111\}$. Using the disjunctive normal form, we add to the circuit one multiqubit Toffoli gate for each

point in $\{001, 011, 110, 111\}$, as follows



Since the first and second gates have opposite controls in the second qubit and identical controls in the first and third qubits (the third and fourth gates have a similar feature), this circuit can be simplified to



This example helps to show how to implement any balanced oracle using $n + 1$ qubits. It is possible to implement the Deutsch-Jozsa algorithm with only n qubits, but this discussion is postponed and fully addressed in Chapter 9.

4.6 Final remarks

There are efficient randomized classical algorithms that solve the Deutsch-Jozsa problem. It is possible to find the correct solution with high probability by consulting the classical oracle a few times. More formally, consider the following randomized algorithm: (1) Select uniformly at random $k \geq 2$ points x_0, ..., x_{k-1} in the domain, repetitions are allowed, (2) if $f(x_0) = \cdots = f(x_{k-1})$, return "$f$ is constant"; otherwise, return "f is balanced". This algorithm returns the output "f is balanced" with certainty because as soon as we obtain two different values after evaluating f, we claim the promise that f is either balanced or constant—it must be balanced. The probability that the output "f is constant" is correct is $1 - 1/2^{k-1}$.[5] The success probability quickly tends to 1, for instance, take $k = 10$, the success probability is at least 99.8%.

[5]To calculate the probability that the output "f is constant" is correct, we start by calculating the probability that "f is constant" is wrong. The output "f is constant" is wrong when f is balanced and $f(x_0) = \cdots = f(x_{k-1})$. The probability that $f(x_0) = \cdots = f(x_{k-1}) = 0$ is $1/2^k$ because each evaluation is independent. Likewise, the probability that $f(x_0) = \cdots = f(x_{k-1}) = 1$ is $1/2^k$. Then, the probability that "f is constant" is wrong is $2/2^k$ because all results equal to 0 and all results equal to 1 are mutually exclusive. Then, the probability that "f is constant" is correct is $1 - 1/2^{k-1}$.

Chapter 5

Bernstein-Vazirani Algorithm

The Bernstein-Vazirani algorithm was presented at a conference in 1993 [8], and the full paper was published in 1997 [9], being the first deterministic quantum algorithm with linear gain over the best deterministic or randomized classical algorithm. The Bernstein-Vazirani algorithm exploits quantum parallelism but has no entanglement at all. This algorithm is described in some books [40, 44, 54].

5.1 Problem formulation

Let $s = s_0...s_{n-1}$ be an unknown n-bit string. Although we don't know s, we have at our disposal a Boolean function $f : \{0,1\}^n \longrightarrow \{0,1\}$ defined as

$$f(x) = s \cdot x = s_0x_0 + ... + s_{n-1}x_{n-1} \mod 2,$$

where $x_0, ..., x_{n-1}$ are the bits of x. Note that f is a linear function and each linear Boolean function is characterized by a specific hidden s. Our goal is to find s by evaluating f without knowing its implementation details. In quantum computing, the operator that implements this function is called *oracle*, as if an oracle reveals $f(x)$ without showing s explicitly, and $f(x)$ can be used to determine s. When we build the circuit of the algorithm, f is implemented by another person, because we don't know s. It is important to understand this; otherwise, we get the impression that we need to know the answer to find the answer, which is absurd.

In the classical version of this problem, we have to consult the *classical oracle* at least n times. In fact, we chose $x = 10...0$ and ask the oracle what is $f(10...0)$. The answer is s_0. Next we choose $x = 010...0$ and ask the oracle what is $f(010...0)$. The answer is s_1. The last query is $f(0...01)$, whose answer is s_{n-1}. This shows that we need to consult the oracle n times and there is no way to reduce this number without introducing an error in the algorithm. In the quantum case, we consult the *quantum oracle* only once, which allows us to find all bits of s, as described below.

In the quantum case, the function $f(x) = s \cdot x$ is implemented using the unitary operator U_f of $n + 1$ qubits, defined as

$$U_f|x\rangle|j\rangle = |x\rangle|j \oplus f(x)\rangle,$$

where $x \in \{0,1\}^n$, j is a bit, and $\oplus$ is the XOR operation or sum modulo 2. This operator uses two registers, with sizes n and 1, respectively. We can use U_f as many times as we wish. However, it is used only once in the Bernstein-Vazirani algorithm.

Summing up, the classical algorithm queries the classical oracle n times using a n-bit classical computer. The quantum algorithm queries the quantum oracle only one time using a $(n+1)$-qubit quantum computer. In the last Section of this Chapter, we show that the algorithm can be implemented on a n-qubit quantum computer.

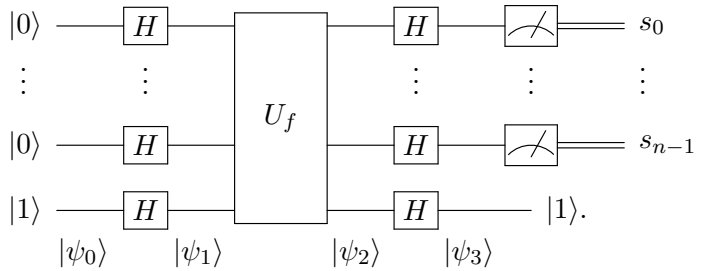
5.2 The algorithm

Algorithm 5.1: Bernstein-Vazirani algorithm

Input: A Boolean function $f : \{0,1\}^n \longrightarrow \{0,1\}$ such that $f(x) = s \cdot x$.
Output: s with probability equal to 1.

1 Prepare the initial state $|0\rangle^{\otimes n}|1\rangle$;
2 Apply $H^{\otimes(n+1)}$;
3 Apply U_f;
4 Apply $H^{\otimes(n+1)}$;
5 Measure the first register in the computational basis.

The Bernstein-Vazirani algorithm is described in Algorithm 5.1 and the circuit is



Note that this circuit is equal to the circuit of the Deutsch-Jozsa algorithm, the only difference is the function used in the Bernstein-Vazirani algorithm, which need not be constant nor balanced. Besides, in the Deutsch-Jozsa algorithm we check whether all outputs are 0 to conclude that the function is constant; otherwise, balanced. In the Bernstein-Vazirani algorithm, each output is valuable to determine the unknown s. The states at the bottom of the circuit are used in the analysis of the algorithm.

5.3 Analysis of the algorithm

After the first step, the state of the qubits is

$$|\psi_0\rangle = |0\rangle^{\otimes n}|1\rangle.$$

After the second step, the state of the qubits is

$$\begin{aligned} |\psi_1\rangle &= (H^{\otimes n}|0\rangle^{\otimes n}) \otimes (H|1\rangle) \\ &= \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle \otimes |-\rangle, \end{aligned}$$

where $|-\rangle = (|0\rangle - |1\rangle)/\sqrt{2}$ and x is written in the decimal notation. After the third step, the state of the qubits is

$$\begin{aligned} |\psi_2\rangle &= U_f|\psi_1\rangle \\ &= \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} U_f|x\rangle \otimes |-\rangle. \end{aligned}$$

To simplify $|\psi_2\rangle$, we use Proposition 3.1 on Page 28, which states that

$$U_f\big(|x\rangle|-\rangle\big) = (-1)^{f(x)}|x\rangle|-\rangle.$$

We obtain

$$|\psi_2\rangle = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} (-1)^{s\cdot x}|x\rangle \otimes |-\rangle.$$

After the fourth step, the state of the qubits is

$$\begin{aligned} |\psi_3\rangle &= H^{\otimes(n+1)}|\psi_2\rangle \\ &= H^{\otimes n}\left(\frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} (-1)^{s\cdot x}|x\rangle\right) \otimes (H|-\rangle). \end{aligned} \tag{5.1}$$

To simplify $|\psi_3\rangle$, we use Proposition 4.2 on Page 35, which states that for any $s = (s_0...s_{n-1})_2$

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

where $s \cdot x = s_0x_0 + \cdots + s_{n-1}x_{n-1} \mod 2$. Then we use $H^2 = I$ to obtain

$$|s\rangle = H^{\otimes n}\left(\frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} (-1)^{s\cdot x}|x\rangle\right).$$

We replace this result in Eq. (5.1) to obtain

$$|\psi_3\rangle = |s\rangle \otimes |1\rangle.$$

The output of the measurement of the first register is $s_0, ..., s_{n-1}$ with probability 1 because $|s\rangle = |s_0\rangle \otimes \cdots \otimes |s_{n-1}\rangle$.

U_f is applied only one time, so we say that a single query was made to the quantum oracle. Note that U_f is applied to a superposition of all vectors of the computational basis, therefore f is evaluated simultaneously at all points in the domain—all n-bit strings. The result of this massive evaluation is a superposition state, which is useless in many a case. Not in the Bernstein-Vazirani algorithm because the application of $H^{\otimes n}$ to the first register at the end reveals s. Before the last step, s was a phase. After, s went into the kernel of a "ket". This makes all the difference because a phase is associated with the probability of obtaining a certain result, while the kernel of the "ket" is associated with the measurement result.

5.4 Bernstein-Vazirani algorithm has no entanglement

The analysis of the entanglement follows at the beginning the same track used in the Deutsch-Jozsa algorithm. From the circuit of the algorithm, we realize that the only operator that creates or destroys entanglement is U_f. Then, it is enough to analyze $|\psi_2\rangle$. There is entanglement in the Bernstein-Vazirani algorithm if and only if $|\psi_2\rangle$ is totally or partially entangled. State $|\psi_2\rangle$ is given by

$$|\psi_2\rangle \;=\; \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} (-1)^{s\cdot x} |x\rangle \otimes |-\rangle.$$

Proposition 4.2 on Page 35 states that for any $s = (s_0 ... s_{n-1})_2$

$$H^{\otimes n}|s\rangle \;=\; \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} (-1)^{s\cdot x} |x\rangle.$$

Then $|\psi_2\rangle$ can be written as

$$|\psi_2\rangle \;=\; \big(H^{\otimes n}|s\rangle\big) \otimes |-\rangle,$$

which can be fully factorized as

$$|\psi_2\rangle \;=\; \big(H|s_0\rangle\big) \otimes \cdots \otimes \big(H|s_{n-1}\rangle\big) \otimes |-\rangle.$$

Note that $|\psi_2\rangle$ has no entanglement at all because it is the Kronecker product of 1-qubit pure states [41, 21].

5.5 Circuit of the oracle

In oracle-based algorithms, the oracle is not implemented by us—it is implemented by someone else. However, it is important to know how it is done in order to understand the whole process. Let us show an example with $n = 4$ and $s = 1011$ that is enough to understand the general case. In this particular case, f is

$$f(x) = s \cdot x = x_0 + x_2 + x_3 \mod 2$$

and the action of U_f on an arbitrary vector of the computational basis $|x\rangle|j\rangle$ is

$$U_f|x_0x_1x_2x_3\rangle|j\rangle = |x_0x_1x_2x_3\rangle|j \oplus f(x)\rangle = |x_0x_1x_2x_3\rangle|j \oplus x_0 \oplus x_2 \oplus x_3\rangle.$$

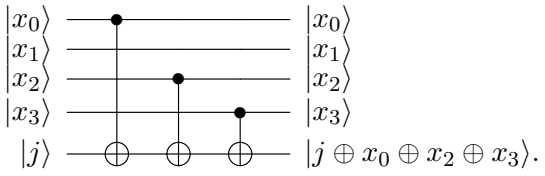
Let CNOT_{04} be the CNOT gate acting on qubits 0 and 4. Without showing qubits 1, 2, and 3, we have

$$\mathrm{CNOT}_{04}|x_0\rangle|j\rangle = |x_0\rangle X^{x_0}|j\rangle = |x_0\rangle|j \oplus x_0\rangle.$$

Note that if we use CNOT_{04}, CNOT_{24}, and CNOT_{34}, we generate the expected result $j \oplus x_0 \oplus x_2 \oplus x_3$ in the second register. Then,

$$U_f \;=\; \mathrm{CNOT}_{34} \cdot \mathrm{CNOT}_{24} \cdot \mathrm{CNOT}_{04},$$

whose circuit is

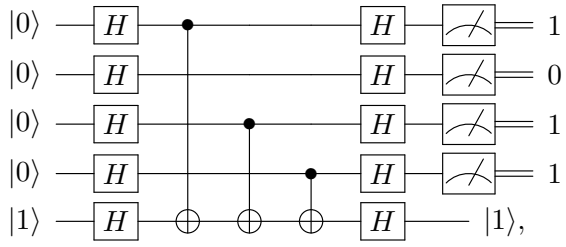


From this example, we can generalize the implementation to an arbitrary $s = s_0 \cdots s_{n-1}$. Just consider

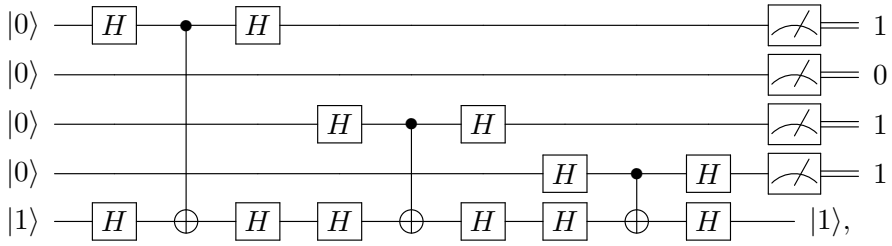
$$U_f \;=\; (\mathrm{CNOT}_{0n})^{s_0}\,(\mathrm{CNOT}_{1n})^{s_1}\,\cdots\,(\mathrm{CNOT}_{n-1,n})^{s_{n-1}}\,,$$

where CNOT_{ij} is controlled by qubit i and the target is qubit j, and $(\mathrm{CNOT}_{ij})^{s_k}$ is the identity operator if $s_k = 0$, and the standard CNOT_{ij} if $s_k = 1$. Note that the order of the CNOT gates is irrelevant. There is a CNOT for each bit 1 of s.

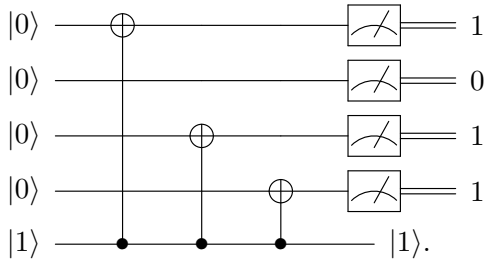
Let us finish the circuit of the Bernstein-Vazirani algorithm for our example. The whole circuit is



which is equivalent to



because $H^2 = I$. Using that $(H \otimes H) \cdot \mathrm{CNOT}_{ij} \cdot (H \otimes H) = \mathrm{CNOT}_{ji}$, the last circuit simplifies to



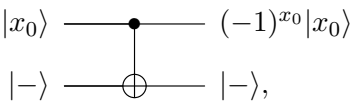
From this example, we can easily derive the generic case, which can always be expressed as CNOT's controlled by the last qubit and the targets are the qubits corresponding to the bits of s that are equal to 1. This simplification helps to understand that the Bernstein-Vazirani algorithm has no entanglement because U_f was multiplied by $H^{\otimes(n+1)}$, which cannot create or destroy entanglement. The depiction of U_f in the circuit of the algorithm gives us the impression that U_f creates entanglement, but that is misleading.

5.6 Economical circuit of the Bernstein-Vazirani algorithm

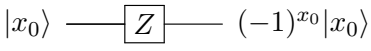
The Bernstein-Vazirani algorithm can be implemented with n qubits instead of $n+1$. Indeed, we have learned that U_f is a product of CNOT's

$$U_f \;=\; (\mathrm{CNOT}_{0n})^{s_0} \cdots (\mathrm{CNOT}_{n-1,n})^{s_{m-1}}\,,$$

where CNOT_{ij} is controlled by qubit i and the target is qubit j, and $(\mathrm{CNOT}_{ij})^{s_k}$ is the identity if $s_k = 0$, and CNOT_{ij} if $s_k = 1$. In the algorithm, just before the action of U_f, the state of the n-th qubit is $|-\rangle$. The circuit that describes the action of CNOT_{0n} is



where we have depicted only the first and the last qubits and we have placed $(-1)^{x_0}$ in the first qubit because this is mathematically allowed. This circuit is equivalent to



We can convert all CNOT's of U_f into Z's and then

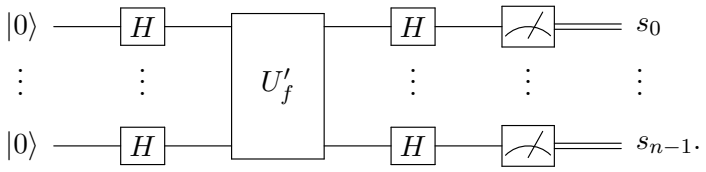
$$U'_f \;=\; Z^{s_0} \otimes \cdots \otimes Z^{s_{n-1}}.$$

Note that if a bit s_i of s is 0, $Z^{s_i} = I$. Then

$$U'_f|x\rangle \;=\; (-1)^{s\cdot x}|x\rangle.$$

Function f is the same as before. What changes is the way we implement f as a unitary operator.

The circuit of the economical version of the Bernstein-Vazirani algorithm is



As before, this circuit simplifies to

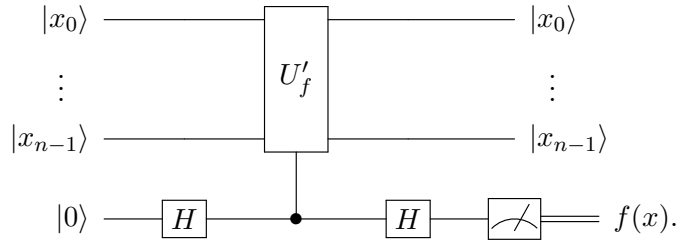
$$H^{\otimes n} U'_f H^{\otimes n} \;=\; X^{s_0} \otimes \cdots \otimes X^{s_{n-1}}$$

because $HZH = X$. Now, it is straightforward to check that there is no entanglement in the Bernstein-Vazirani algorithm. U'_f is not a genuine n-qubit gate, but instead is the tensor product of n 1-qubit gates. The depiction of the circuit is misleading.

Consulting the oracle

In the non-economical circuit, if the input to the first register of U_f is $|x\rangle$, U_f returns $f(x) = (-1)^{s\cdot x}$ when we perform a measurement of the last qubit. In the economical circuit, we obtain

$f(x)$ using the circuit

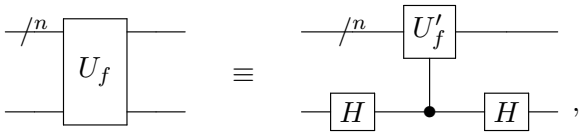


In fact, the steps of this circuit are

$$|x\rangle|0\rangle \xrightarrow{I\otimes H} |x\rangle|+\rangle \xrightarrow{U'_f} \frac{|x\rangle|0\rangle + (-1)^{f(x)}|x\rangle|1\rangle}{\sqrt{2}} \xrightarrow{I\otimes H} |x\rangle|f(x)\rangle,$$

where I is the 2^n-dimensional identity operator. To calculate the last step, there are only two cases when we fix x: Either $f(x) = 0$ or $f(x) = 1$. Firstly, we suppose that $f(x) = 0$, then $I \otimes H$ is applied to $|x\rangle|+\rangle$ resulting in $|x\rangle|f(x)\rangle$, and secondly, we suppose that $f(x) = 1$, then $I \otimes H$ is applied to $|x\rangle|-\rangle$ resulting in $|x\rangle|f(x)\rangle$. After performing a measurement of the last qubit in the computational basis, we obtain $f(x)$ with probability 1.

We have just shown the following circuits are equivalent:



where the notation $/^n$ on a wire represents a n-qubit register.

Chapter 6

Simon's Problem

Simon's problem was presented at a conference in 1994 [60] together with Shor's algorithms [57], and the full paper was published in 1997 [61]. It is a quantum algorithm exponentially faster than the best deterministic or randomized equivalent classical algorithm. It is a remarkable but underestimated scientific contribution to quantum computing. Simon's algorithm exploits not only quantum parallelism but also maximal entanglement. This algorithm and its generalization are described in some books [30, 40, 44, 54, 67] and papers [13, 42, 68].

6.1 Problem formulation

Let $f : \{0,1\}^n \longrightarrow \{0,1\}^n$ be a n-bit input to n-bit output function with the following property: There is a nonzero hidden bit string $s \in \{0,1\}^n$ such that

$$f(x) = f(y) \iff x \oplus y \in \{0, s\}$$

for all $x, y \in \{0,1\}^n$. This means that each point in the image is the image of exactly two points in the domain, that is, the point $f(x)$ in the image has associated x and $x \oplus s$ in the domain because $f(x) = f(x \oplus s)$ for all $x \in \{0,1\}^n$. Because of that, function f is in the set of two-to-one functions. Consider the following computational problem: Determine s by querying f as few as possible.

Our goal is to find s by evaluating f without knowing its implementation details. In quantum computing, the operator that implements this function is called *oracle*, as if an oracle reveals $f(x)$ without showing s explicitly, and $f(x)$ can be used to determine s, but to evaluate f only once is not enough. When we build the circuit of the algorithm, the part associated with f is implemented by another person, because we don't know s.

In the classical version of this problem, we have to consult a *classical oracle*, and to determine s with high probability, the number of queries to the function f grows as an exponential function on the number of bits n if we use a classical computer. Indeed, a naive deterministic algorithm would be to calculate $f(x')$ for a fixed x', then systematically search for x in the domain such that $f(x) = f(x')$. We need 2^n evaluations in the worst case. It is better to pick x and x' uniformly at random and then check whether $f(x) = f(x')$. This method requires $\Omega(\sqrt{2^n})$ evaluations to achieve success probability $\Omega(1)$. The proof is the same as in the two-to-one collision problem [12] or in the *birthday paradox* [16].

In the quantum case, f is implemented using the unitary operator U_f of $2n$ qubits, defined as

$$U_f|x\rangle|y\rangle = |x\rangle|y \oplus f(x)\rangle,$$

where x, y, and $f(x)$ are n-bit strings, and $\oplus$ is the bitwise XOR operation or bitwise sum modulo 2. This operator uses two registers each one with n qubits. U_f is a 2^{2n}-dimensional permutation matrix. The proof that U_f is unitary is an extension of the proof presented in Proposition 3.1 on Page 28.

Simon's algorithm has two parts. The quantum part returns a n-bit string x obeying $x \cdot s = 0$, where

$$x \cdot s = x_0 s_0 + \cdots + x_{n-1} s_{n-1} \mod 2.$$

To know such x is not enough to determine s. It is necessary to run the quantum part many times, collect many bit strings x obeying $x \cdot s = 0$, and then run the classical part of the algorithm, which reveals s with probability greater than 1/2.

6.2 The algorithm

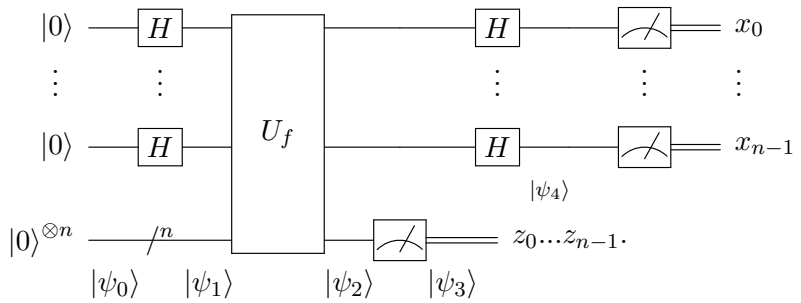
Algorithm 6.1: Simon's algorithm

Input: Function $f : \{0,1\}^n \longrightarrow \{0,1\}^n$ with the promise that
$f(x) = f(y) \iff x \oplus y \in \{0, s\}$.

Output: s with probability greater than 1/2.

1 Run the quantum part $n-1$ times (Algorithm 6.2, assume outputs $x^{(1)}, ..., x^{(n-1)}$);
2 Solve the system of linear equations $\{x^{(1)} \cdot s \equiv 0, ..., x^{(n-1)} \cdot s \equiv 0\} \mod 2$ (assume solution for $s_1, ..., s_{n-1}$);
3 Take $s_0 = 0$;
4 If $f(s) = f(0)$ then return $0s_1...s_{n-1}$; otherwise, return $1s_1...s_{n-1}$.

Simon's algorithm is described in Algorithm 6.1 and the quantum part is described in Algorithm 6.2. The circuit of the quantum part is



The states at the bottom of the circuit are used in the analysis of the algorithm. They describe the state of the qubits after each step. Note that state $|\psi_4\rangle$ refers to the first register only. The notation "$/^n$" over a wire denotes that it is a n-qubit register.

Algorithm 6.2: Quantum part of Simon's algorithm

Input: A black box U_f implementing function $f : \{0,1\}^n \longrightarrow \{0,1\}^n$ with the promise that $f(x) = f(y) \iff x \oplus y \in \{0, s\}$.

Output: Point $x \in \{0,1\}^n$ such that $x \cdot s = 0$.

1 Prepare the initial state $|0\rangle^{\otimes n}|0\rangle^{\otimes n}$;
2 Apply $H^{\otimes n}$ to the first register;
3 Apply U_f;
4 Measure the second register in the computational basis (assume output $z_0...z_{n-1}$);
5 Apply $H^{\otimes n}$ to the first register;
6 Measure the first register in the computational basis.

6.3 Analysis of the quantum part

After the first step, the state of the qubits is

$$|\psi_0\rangle = |0\rangle^{\otimes n}|0\rangle^{\otimes n}.$$

After the second step, the state of the qubits is

$$\begin{aligned} |\psi_1\rangle &= \big(H|0\rangle\big)^{\otimes n} \otimes |0\rangle^{\otimes n} \\ &= \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle \otimes |0\rangle^{\otimes n}, \end{aligned}$$

where x is written in the decimal notation. After the third step, the state of the qubits is

$$\begin{aligned} |\psi_2\rangle &= U_f|\psi_1\rangle \\ &= \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} U_f(|x\rangle \otimes |0\cdots 0\rangle). \end{aligned}$$

To simplify $|\psi_2\rangle$, we use the definition of U_f to obtain

$$|\psi_2\rangle = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle \otimes |f(x)\rangle,$$

because $(0...0) \oplus f(x)$ (bitwise XOR) is $f(x)$. The fourth step is a measurement of each qubit of the second register, which we assume has returned $z_0...z_{n-1}$. State $|\psi_2\rangle$ collapses to a superposition of only two terms because there are only two points in the domain such that $f(x) = z_0...z_{n-1}$. Let x' be one of those points. Then, $f(x') = f(x' \oplus s) = z_0...z_{n-1}$ and state $|\psi_3\rangle$ is

$$|\psi_3\rangle = \left(\frac{|x'\rangle + |x' \oplus s\rangle}{\sqrt{2}}\right) \otimes |z_0...z_{n-1}\rangle.$$

Note that we have renormalized state $|\psi_3\rangle$ as demanded by the measurement postulate. Point x' is unknown. It is selected uniformly at random among the domain points. The information we wish to acquire, s, is hidden because $x' \oplus s$ is a random point in the domain.

In the fifth step, we only consider the first register. After applying $H^{\otimes n}$ to the state of the first register, we obtain

$$|\psi_4\rangle \;=\; \frac{1}{\sqrt{2}}\left(H^{\otimes n}|x'\rangle + H^{\otimes n}|x'\oplus s\rangle\right).$$

To simplify $|\psi_4\rangle$, we use Proposition 4.2 on Page 35, which states that for any $x' = (x'_0...x'_{n-1})_2$

$$H^{\otimes n}|x'\rangle \;=\; \frac{1}{\sqrt{2^n}}\sum_{x=0}^{2^n-1}(-1)^{x'\cdot x}|x\rangle,$$

where $x'\cdot x = x'_0x_0 + \cdots + x'_{n-1}x_{n-1} \mod 2$. Then

$$|\psi_4\rangle \;=\; \frac{1}{\sqrt{2^{n+1}}}\sum_{x=0}^{2^n-1}\left((-1)^{x'\cdot x} + (-1)^{(x'\oplus s)\cdot x}\right)|x\rangle.$$

Now we use the fact that

$$\begin{aligned}(x'\oplus s)\cdot x &= (x'_0+s_0)x_0 + \cdots + (x'_{n-1}+s_{n-1})x_{n-1} \mod 2\\ &= (x'_0x_0 + \cdots + x'_{n-1}x_{n-1}) + (s_0x_0 + \cdots + s_{n-1}x_{n-1}) \mod 2\\ &= (x'\cdot x) + (s\cdot x) \mod 2.\end{aligned}$$

We have used that $x'_0 \oplus s_0 = x'_0 + s_0 \mod 2$. State $|\psi_4\rangle$ simplifies to

$$|\psi_4\rangle \;=\; \frac{1}{\sqrt{2^{n+1}}}\sum_{x=0}^{2^n-1}(-1)^{x'\cdot x}\left(1+(-1)^{s\cdot x}\right)|x\rangle.$$

Now we use

$$1+(-1)^{s\cdot x} = \begin{cases}2, & \text{if } s\cdot x = 0,\\ 0, & \text{otherwise,}\end{cases}$$

to obtain

$$|\psi_4\rangle \;=\; \frac{1}{\sqrt{2^{n-1}}}\sum_{\substack{x=0\\ s\cdot x=0}}^{2^n-1}(-1)^{x'\cdot x}|x\rangle.$$

The sum is over x such that $x\cdot s = 0$. Then, the output of the measurement of the first register is x such that $x\cdot s = 0$ with probability $|(-1)^{x'\cdot x}|^2 = 1$. When we run the quantum part of Simon's algorithm, we obtain partial information about s. This means that we have to run the quantum part many times to collect enough information to determine s.

Note that the probability of obtaining a specific x such that $x\cdot s = 0$ is $1/2^{n-1}$, which is the square of the absolute value of the amplitude of state $|x\rangle$ in $|\psi_4\rangle$. The distribution is uniform over all of n-bit strings x that satisfy $x\cdot s = 0$. On the other hand, the probability of obtaining an arbitrary x such that $x\cdot s = 0$ is 1, or equivalently the probability of obtaining x such that $x\cdot s \neq 0$ is 0. That is, we are 100% sure that the output x satisfies $x\cdot s = 0$. We learn partial information about s unless $x = (0...0)_2$.

Point x', which obeys $f(x') = z$, plays no role in the final result nor in the final calculation of the success probability. This is good news because x' was hiding s at some point. After the fifth step, x' becomes harmless.

6.4 Analysis of the classical part

Each time we run the quantum part of Simon's algorithm, the output is a n-bit string x such that $x \cdot s = 0$. Suppose we have run two times and the outputs are x and x'. This means that we have obtained a homogeneous system of linear equations

$$\begin{aligned} x_0 s_0 + \cdots + x_{n-1} s_{n-1} &\equiv 0 \mod 2, \\ x'_0 s_0 + \cdots + x'_{n-1} s_{n-1} &\equiv 0 \mod 2, \end{aligned}$$

where $s_0, ..., s_{n-1}$ are the variables (unknowns) and x, x' are known binary coefficients. Let us calculate probability $p(2)$ that the system is independent. The probability that the first equation is nontrivial is

$$p_1 = 1 - \frac{1}{2^n}$$

because there are 2^n strings x and only one is 0. To calculate the probability that the second equation is independent, we think that x is a n-dimensional vector in a binary vector space with 2^n vectors. The subspace spanned by x has two vectors, x itself and the null vector. There are $2^n - 2$ vectors that are linearly independent of x. Then, the probability p_2 that the second equation is independent is

$$p_2 = 1 - \frac{2}{2^n}.$$

Then, probability $p(2)$ that the two equations are independent is

$$p(2) = p_1 p_2 = \left(1 - \frac{1}{2^n}\right)\left(1 - \frac{2}{2^n}\right).$$

The probability that the next equation added to the system is independent is calculated as follows. Vectors x and x' span a subspace with four vectors: x, x', $x \oplus x'$, and the null vector. There are $2^n - 4$ vectors that are linearly independent of x and x'. The probability p_3 that the third equation is independent is

$$p_3 = 1 - \frac{2^2}{2^n}.$$

Then, probability $p(3)$ that the three equations are independent is

$$p(3) = p_1 p_2 p_3 = \left(1 - \frac{1}{2^n}\right)\left(1 - \frac{2}{2^n}\right)\left(1 - \frac{2^2}{2^n}\right).$$

We proceed in this fashion until we obtain $n - 1$ independent equations with probability

$$p(n-1) = \prod_{i=1}^{n-1} p_i = \prod_{i=0}^{n-2} \left(1 - \frac{2^i}{2^n}\right).$$

This product is hard to calculate. The goal now is to find a nontrivial lower bound. If we expand the product we obtain

$$\left(1 - \frac{1}{2^n}\right) \cdots \left(1 - \frac{2^{n-2}}{2^n}\right) = 1 - \sum_{i=0}^{n-2} \frac{2^i}{2^n} + \cdots .$$

The sum is calculated using the geometric series yielding $(1/2 - 1/2^n)$. The $(\cdots)$ part has higher-order terms $(1/(2^n)^2, 1/(2^n)^3, ...)$, and the next Proposition shows that it is positive. Then,

$$p(n-1) \geq \frac{1}{2} + \frac{1}{2^n}.$$

Proposition 6.1. Let $n \geq 2$ be an integer. Then

$$\prod_{i=0}^{n-2} \left(1 - \frac{2^i}{2^n}\right) \geq \frac{1}{2} + \frac{1}{2^n}.$$

Proof. By induction on n. The base case follows after replacing n with 2. Let's prove the induction step. The left-hand expression for $n+1$ is

$$\prod_{i=0}^{n-1} \left(1 - \frac{2^i}{2^{n+1}}\right) = \left(1 - \frac{2^0}{2^{n+1}}\right) \prod_{i=1}^{n-1} \left(1 - \frac{2^i}{2^{n+1}}\right).$$

By manipulating the dummy index i, we obtain

$$\prod_{i=0}^{n-1} \left(1 - \frac{2^i}{2^{n+1}}\right) = \left(1 - \frac{1}{2^{n+1}}\right) \prod_{i=0}^{n-2} \left(1 - \frac{2^i}{2^n}\right).$$

Let us assume that the inequality is true for n. Then

$$\prod_{i=0}^{n-1} \left(1 - \frac{2^i}{2^{n+1}}\right) \geq \left(1 - \frac{1}{2^{n+1}}\right) \left(\frac{1}{2} + \frac{1}{2^n}\right).$$

Expanding the right-hand side, we obtain

$$\prod_{i=0}^{n-1} \left(1 - \frac{2^i}{2^{n+1}}\right) \geq \frac{1}{2} + \frac{1}{2^{n+1}} + \frac{1}{2^{n+1}} \left(\frac{1}{2} - \frac{1}{2^n}\right) \geq \frac{1}{2} + \frac{1}{2^{n+1}}.$$

This completes the proof. □

We are not done yet because with $n-1$ independent equations we determine $n-1$ bits of s. The missing bit s_i is determined by guessing, for instance, by assuming that $s_i = 0$ and then we use the classical oracle and ask whether $f(s) = f(0)$? If true, we have already found s; otherwise, we set $s_i = 1$. The cost of running the classical part is basically the cost of solving a system of $n-1$ linear equations with n variables, which is the cost $O(n^2)$ of inverting a $n \times n$ matrix.

The total cost of the algorithm is $n-1$ calls of U_f and one call of f plus $O(n^2)$ steps to solve the system of linear equations. The success probability is greater than $1/2$.

6.5 Analysis of the entanglement

From the circuit of the algorithm, we realize that the only operator that creates or destroys entanglement is U_f. Then, it is enough to analyze $|\psi_2\rangle$ or $|\psi_3\rangle$. It is simpler to analyze $|\psi_3\rangle$. There is entanglement in Simon's algorithm if and only if $|\psi_3\rangle$ is totally or partially entangled. The state of the first register of $|\psi_3\rangle$ is

$$|\psi\rangle \;=\; \frac{|x\rangle + |x \oplus s\rangle}{\sqrt{2}},$$

where x is a random n-bit string and s is a fixed nonzero n-bit string. If $s = 1...1$ and $x = 1...1$, $|\psi\rangle$ is the well-known Greenberger–Horne–Zeilinger state of n qubits, defined as

$$|\mathrm{GHZ}\rangle \;=\; \frac{|0\cdots0\rangle + |1\cdots1\rangle}{\sqrt{2}}.$$

It is known that the GHZ state is maximally entangled.[1] If $s = 1...1$, state $|\psi\rangle$ is non-biseparable[2] for any x because

$$|\psi\rangle \;=\; X^{x_0} \otimes \cdots \otimes X^{x_{n-1}} |\mathrm{GHZ}\rangle,$$

and $X^{x_0} \otimes \cdots \otimes X^{x_{n-1}}$ does not create or destroy entanglement.

On the other hand, we can factor state $|\psi\rangle$ for each bit 0 of s. Suppose that $s = 01...1$, then

$$|\psi\rangle \;=\; |x_0\rangle \otimes \frac{|x_1...x_{n-1}\rangle + |\bar{x}_1...\bar{x}_{n-1}\rangle}{\sqrt{2}},$$

where $\bar{x}_i = x_i \oplus 1$. This state is not maximally entangled but it is still partially entangled if $n > 2$. If $s = 0...01$, then

$$|\psi\rangle \;=\; |x_0\rangle \otimes \cdots |x_{n-2}\rangle \otimes \frac{|x_{n-1}\rangle + |\bar{x}_{n-1}\rangle}{\sqrt{2}},$$

which has no entanglement at all.

Summing up, if the Hamming weight of s is larger than 1, there is entanglement in Simon's algorithm. The amount of entanglement increases with the Hamming weight of s and the state of the first register before the measurement is maximally entangled if the Hamming weight of s is n.

6.6 Circuit of the oracle

In oracle-based algorithms, the oracle is not implemented by us—it is implemented by someone else. However, it is important to know how it is done in order to understand the whole process. Let us show an example with $n = 3$ and $s = 110$ that is enough to understand the general case.

[1] https://en.wikipedia.org/wiki/Greenberger-Horne-Zeilinger_state

[2] A pure state $|\psi\rangle$ of n qubits is called biseparable, if one can find a partition of the qubits in two registers A and B such that $|\psi\rangle = |\psi_A\rangle \otimes |\psi_B\rangle$.

Let us take f as the following two-to-one function

$x_0\, x_1\, x_2$	$f(x)$
0 0 0 1 1 0	000
0 0 1 1 1 1	001
0 1 0 1 0 0	010
0 1 1 1 0 1	100

To build the circuit we need to write down the explicit 3-output truth table, which is

$x_0\, x_1\, x_2$	$f_0(x)$	$f_1(x)$	$f_2(x)$
0 0 0	0	0	0
0 0 1	0	0	1
0 1 0	0	1	0
0 1 1	1	0	0
1 0 0	0	1	0
1 0 1	1	0	0
1 1 0	0	0	0
1 1 1	0	0	1

Note that $f(x) = f_0(x)f_1(x)f_2(x)$, where f_0 to f_2 are Boolean functions. The truth table of f_0 is obtained by considering only the first column of the output, and the truth tables of f_1 and f_2 by considering the second and third columns, respectively. Now we focus on all bits 1 in the first column of the output denoted by $f_0(x)$. There are two of them, corresponding to inputs 011 and 101. We add to the circuit two multiqubit Toffoli gates, the first with controls activated by 011 and the second activated by 101, with the target on the 4th qubit, as shown in Fig. 6.1. Then, we focus on all bits 1 in the second column of the output denoted by $f_1(x)$. There are two of them, corresponding to the inputs 010 and 100. The multiqubit Toffoli gates are activated by 010 and 100, respectively, with the target on the 5th qubit, as shown in Fig. 6.1. The last column, denoted by $f_2(x)$, requires multiqubit Toffoli gates activated by 001 and 111 with the target on the 6th qubit, as shown in Fig. 6.1.

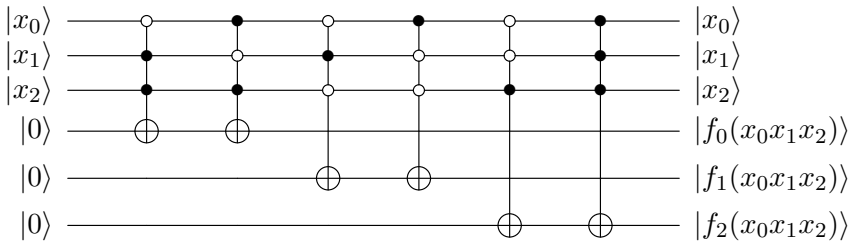


Figure 6.1: Oracle of Simon's algorithm.

The only trivial simplification that can be immediately seen is that the first two multiqubit Toffoli gates can be simplified into only one Toffoli gate with empty control on qubit 2, full control on qubit 3 and target on qubit 4.

6.7 Final remarks

The formulation of Simon's problem in the original paper [61] is slightly different from the one presented here. Simon posed the problem of determining whether f is one-to-one (injective) or a special kind of two-to-one characterized by a n-bit string s such that $f(x') = f(x)$ if and only if $x' = x \otimes s$. In the latter case, we have to find s. This formulation goes along the line of the Deutsch-Jozsa algorithm, in which we have the promise that the oracle is either balanced or constant. We have to determine which is the case. Note that if we run the quantum part of Simon's algorithm with a one-to-one function, the output is a random n-bit string. Simon used this fact to prove that there exists an algorithm for a quantum Turing machine that solves Simon's problem with zero error probability in expected time $O(nT_f(n) + G(n))$, where $T_f(n)$ is the time required to compute $f(x)$, and $G(n)$ is the time required to solve an $n \times n$ linear system of equations over $\mathbb{Z}_2$.

Chapter 7

Shor's Algorithm for Factoring Integers

Shor's algorithms were presented at a conference in 1994 [57]. The full paper was published in 1997 [58], and reviewed in 1999 [59]. It describes two quantum algorithms for integer factoring and discrete logarithm that run in polynomial time. The best-known classical algorithms run in sub-exponential time. Shor's algorithms exploit not only quantum parallelism but also entanglement, being a remarkable and celebrated scientific contribution to quantum computing. The algorithm for factoring integers is the focus of this Chapter and is described in many books [27, 30, 40, 45, 44, 54, 56, 63, 67].

7.1 Problem formulation

Let N be a composite natural number. The computational problem consists in finding a nontrivial factor of N. If N is composite, there exist natural numbers p_1 and p_2 such that $N = p_1 p_2$, where $1 < p_1, p_2 < N$. The goal is to find p_1 and then p_2 is obtained by calculating N/p_1.

Note that if we have an algorithm that finds a nontrivial factor of N efficiently then we are able to find all prime factors of N efficiently because the number of prime factors of N is at most $\log_2 N$.

7.2 Preliminaries on number theory

Although factoring integers is the main interest because it has a big impact in breaking cryptography methods such as RSA and Diffie-Hellman key exchange, we can turn our attention to the problem of finding the multiplicative order of a natural number a modulo N because if we solve the latter problem efficiently then we also solve the factoring problem efficiently.

In number theory, the problem of finding the multiplicative order of a natural number a modulo N aims to determine the smallest positive integer r such that

$$a^r \equiv 1 \mod N.$$

For example, let $N = 21$, which is the largest composite number factored on a quantum computer so far using Shor's algorithm [62]. Now pick at random a number a such that $1 < a < N$. Let

us say $a = 2$. Then, we obtain the following sequence if we keep multiplying each line by a and simplifying using modular arithmetic:

$$\begin{array}{llll} a \equiv 2 & \mod 21 & a^7 \equiv 2 & \mod 21 \\ a^2 \equiv 4 & \mod 21 & a^8 \equiv 4 & \mod 21 \\ a^3 \equiv 8 & \mod 21 & a^9 \equiv 8 & \mod 21 \\ a^4 \equiv 16 & \mod 21 & a^{10} \equiv 16 & \mod 21 \\ a^5 \equiv 11 & \mod 21 & a^{11} \equiv 11 & \mod 21 \\ a^6 \equiv 1 & \mod 21 & a^{12} \equiv 1 & \mod 21 \end{array} \quad \cdots$$

The multiplicative order of 2 modulo 21 is 6 because 6 is the smallest positive integer such that $2^6 \equiv 1 \mod 21$, as we see at the end of the first column. If we continue this sequence, the results 2, 4, and so on will repeat again and again because $a^{r+1} \equiv a^1 \equiv 2$, $a^{r+2} \equiv a^2 \equiv 4$, and so on. Then, we have a r-periodic sequence.

If r is even, then

$$\begin{aligned} \left(a^{\frac{r}{2}} + 1\right)\left(a^{\frac{r}{2}} - 1\right) &\equiv a^r - 1 \mod N \\ &\equiv 0 \mod N. \end{aligned}$$

We find two numbers $a^{r/2} + 1$ and $a^{r/2} - 1$ whose product is a multiple of N. If those numbers are not zero nor multiple of N, then $a^{r/2} + 1$ and $a^{r/2} - 1$ must have factors whose product is N. We conclude that in this case $\gcd\left(a^{r/2} + 1, N\right) > 1$ and $\gcd\left(a^{r/2} - 1, N\right) > 1$, where gcd is the *greatest common divisor*.

For example, when $a = 2$ and $r = 6$, we have $a^{r/2} + 1 = 9$ and $a^{r/2} - 1 = 7$, and in both cases the gcd returns a nontrivial factor of 21. The method fails when $a = 5$ because the multiplicative order of 5 modulo 21 is $r = 6$ and $a^{r/2} + 1 = 126$ and $a^{r/2} - 1 = 124$. In the first case, 126 is a multiple of 21, and in the second case, $\gcd(124, 21) = 1$.

In Shor's algorithm, we start with the number N, then we pick uniformly at random a number a such that $1 < a < N$. Before calculating the order of a, we check whether $\gcd(a, N) > 1$ because (1) if $\gcd(a, N) > 1$ then there is no r such that $a^r \equiv 1 \mod N$ and (2) if $\gcd(a, N) > 1$ then $\gcd(a, N)$ is a nontrivial factor of N, and then we are done. To better understand what is going on here, we split the set of numbers $\{1, ..., N\}$ into two subsets:

$$\begin{aligned} S_1 &= \{a : 1 < a < N \text{ and } \gcd(a, N) > 1\}, \\ S_2 &= \{a : 1 \leq a < N \text{ and } \gcd(a, N) = 1\}. \end{aligned}$$

In our example with $N = 21$, we have

$$\begin{aligned} S_1 &= \{3, 6, 7, 9, 12, 14, 15, 18\}, \\ S_2 &= \{1, 2, 4, 5, 8, 10, 11, 13, 16, 17, 19, 20\}. \end{aligned}$$

When we pick a number a at random, if $a \in S_1$, we quickly find a factor of N by calculating $\gcd(a, N)$ using the Euclidean algorithm.[1] The question now is what is the cardinality of S_1? Is it larger than the cardinality of S_2? To answer this question we use the following facts about S_2 for an arbitrary N, which is denoted by $\mathbb{Z}_N^*$ in number theory [25, 46]:

[1] https://en.wikipedia.org/wiki/Euclidean_algorithm

Fact 1 $\mathbb{Z}_N^*$ is a finite multiplicative group modulo N.

Fact 2 The cardinality of $\mathbb{Z}_N^*$ is Euler's totient function $\varphi(N)$.

Fact 1 is the theoretical basis that guarantees that there exists r such that $a^r \equiv 1 \mod N$ for any $a \in \mathbb{Z}_N^*$. It also guarantees that the function

$$f(\ell) = a^\ell \mod N$$

is r-periodic. On the other hand, Fact 2 is the definition of Euler's totient function $\varphi(N)$, which has been widely studied in number theory. It is known that $\varphi(N)$ is always nearly N (Hardy and Wright [25]). Since the cardinality of S_1 is $N - \varphi(N) - 1$, the probability of selecting $a \in S_1$ is much smaller than the probability of selecting $a \in \mathbb{Z}_N^*$ when N is large.

For instance, assume that $N = p_1 p_2$, where p_1 and p_2 are prime numbers such that the number of figures of p_1 is close to the number of figures of the integer part of $\sqrt{N}$. The same is true for p_2. This is the most interesting case in cryptography because this is the hardest case for classical factoring algorithms. Those primes usually have 1024 bits, which is close to 308 decimal figures. For this case, we have

$$S_1 = \{p_1, 2p_1, ..., (p_2 - 1)p_1, p_2, 2p_2, ..., (p_1 - 1)p_2\}$$

and the cardinality of S_1 is $p_1 + p_2 - 2$, which is close to $2\sqrt{N}$. For large N, we see that there are more odds of picking a from $\mathbb{Z}_N^*$.

Not all a's in $\mathbb{Z}_N^*$ are good because the order of a may be odd or $a^{r/2} + 1 \equiv 0 \mod N$. If we select a bad a, we have to discard it and pick at random another one. The question now is how many a's in $\mathbb{Z}_N^*$ have even order and at the same time $a^{r/2} + 1 \not\equiv 0 \mod N$?

Fact 3 (Theorem A4.13 of [45]) Suppose $N = p_1^{\alpha_1} \cdots p_m^{\alpha_m}$ is the prime factorization of an odd composite positive integer. Let a be chosen uniformly at random from $\mathbb{Z}_N^*$, and let r be the order of a modulo N. Then prob(r is even and $a^{r/2} + 1 \not\equiv 0 \mod N$) $\geq 1 - 1/2^m \geq 3/4$.

The theorem above states that the probability of selecting a good a is at least 3/4. Note that the case $a^{r/2} - 1 \equiv 0 \mod N$ never happens because by definition r is the smallest integer such that $a^r - 1 \equiv 0 \mod N$. Then, if r is even and $a^{r/2} + 1 \not\equiv 0 \mod N$, we have $\gcd\big(a^{r/2} + 1, N\big) > 1$ and at the same time $\gcd\big(a^{r/2} - 1, N\big) > 1$; the results of these gcd's are nontrivial factors of N.

A nontrivial divisor of N is readily obtained as soon as we find a nontrivial integer solution to the equation

$$x^2 \equiv 1 \mod N,$$

such that $x \not\equiv \pm 1 \mod N$. The goal of Shor's algorithm is to find x by running a quantum part that outputs an even r such that $x^{r/2} \not\equiv \pm 1 \mod N$.

7.3 Quantum operator for modular exponentiation

The unitary operator $U_N^{(a)}$, which calculates the exponentiation of integer a modulo N, is defined as

$$U_N^{(a)}|\ell\rangle|y\rangle = |\ell\rangle\Big|y \oplus \big(a^\ell \mod N\big)\Big\rangle, \text{ for } 0 \leq \ell < q,\ 0 \leq y < 2^n,$$

where $n = \lceil \log_2 N \rceil$, $\oplus$ is the bitwise XOR operation or bitwise sum modulo 2, and q is the smallest power of 2 such that $q > N^2$. $U_N^{(a)}$ acts on two registers with sizes $\log_2 q$ and n, and it is a permutation matrix of dimension $2^n q$. The inputs of the algorithm, a and N, come through $U_N^{(a)}$. It replaces the oracle U_f in Simon's algorithm, but $U_N^{(a)}$ is no oracle because it is our task to implement it. In Shor's algorithm, $U_N^{(a)}$ is used many times with a different a each time. The proof that $U_N^{(a)}$ is unitary is an extension of the proof presented in Proposition 3.1 on Page 28.

To implement $U_N^{(a)}$ efficiently, it is necessary to use the *repeated squaring method*, which is an efficient algorithm to calculate modular exponentiation. For instance, if we want to calculate 3^{16} mod 7, we would naively multiply $3 \times 3 \times ... \times 3$ sixteen times, obtain a large number, and then calculate the remainder after dividing by 7. In the exponentiation by squaring, we calculate the square of 3 modulo 7, then we calculate the square of the result modulo 7, and so on four times, which requires a logarithmic number of multiplications and each result is never too large. When we calculate $a^\ell \mod N$, ℓ is not a power of 2 in general, but the repeated squaring method still can be used. Using this method, it is possible to find an efficient circuit of $U_N^{(a)}$ by converting classical irreversible circuits into reversible ones [38, 44, 48].

7.4 Fourier transform and its inverse

The q-dimensional Fourier Transform F_q is a linear operator whose action on the computational basis is

$$F_q|k\rangle = \frac{1}{\sqrt{q}} \sum_{\ell=0}^{q-1} \omega^{k\ell} |\ell\rangle,$$

where $0 \leq k < q$ and $\omega = \mathrm{e}^{2\pi \mathrm{i}/q}$. Note that the (k,ℓ)-entry of F_q is

$$\left(F_q\right)_{k\ell} = \frac{\omega^{k\ell}}{\sqrt{q}}.$$

Then, F_q is a symmetric matrix. To find $F_q^\dagger$, we simply take the complex conjugate of each entry, which is $\omega^{-k\ell}/\sqrt{q}$ because the complex conjugate of ω is ω^{-1}. Then

$$F_q^\dagger|k\rangle = \frac{1}{\sqrt{q}} \sum_{\ell=0}^{q-1} \omega^{-k\ell} |\ell\rangle,$$

where $0 \leq k < q$.

Let us show that F_q is unitary. Using the definitions of F_q and $F_q^\dagger$, we have

$$\begin{aligned}\left\langle k' \right| F_q^\dagger F_q |k\rangle &= \left(\frac{1}{\sqrt{q}} \sum_{\ell'=0}^{q-1} \omega^{-k'\ell'} \left\langle \ell' \right| \right) \left(\frac{1}{\sqrt{q}} \sum_{\ell=0}^{q-1} \omega^{k\ell} |\ell\rangle \right) \\ &= \frac{1}{q} \sum_{\ell=0}^{q-1} \omega^{(k-k')\ell}.\end{aligned}$$

Now we use the closed-form formula for the geometric series with q terms, which is

$$\sum_{k=0}^{q-1} s^k = \frac{1-s^q}{1-s},$$

if $s \neq 1$. When $s = 1$, the left-hand sum is equal to q. In our case $s = \omega^{(k-k')}$. From the definition of ω, we have $\omega^q = 1$, and then $\omega^{(k-k')q} = 1$. Combining those results, we obtain

$$\frac{1}{q}\sum_{\ell=0}^{q-1}\omega^{(k-k')\ell} = \begin{cases}1, & \text{if } k = k',\\ 0, & \text{otherwise.}\end{cases}$$

We have just shown that

$$\left\langle k'\middle|F_q^\dagger F_q|k\right\rangle = \delta_{kk'},$$

that is, $F_q^\dagger F_q = I$.

In Shor's algorithm, given N, q is the smallest power of 2 such that $q > N^2$. F_q is applied only to the first register. The number of qubits of the first register is $\log_2 q$, which is at most $2n$ and at least $2n-1$, while the number of qubits of the second register is exactly $n = \lceil \log_2 N \rceil$. F_q plays a role similar to the Hadamard gates at the end of Simon's algorithm. The circuit of F_q in terms of CNOT and one-qubit gates is described in Sec. 7.8.

7.5 The algorithm

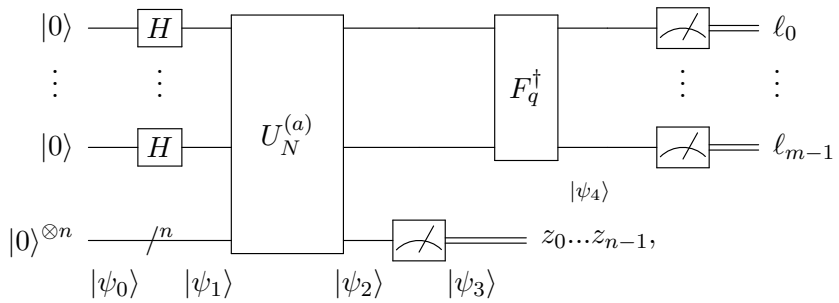
Algorithm 7.1: Shor's algorithm

Input: Composite integer N.

Output: A nontrivial factor of N.

1 If N is even, return 2; otherwise, continue;
2 If N is a power of some prime number p, return p; otherwise, continue;
3 Pick uniformly at random an integer a such that $1 < a < N$;
4 If $\gcd(a, N) > 1$, return $\gcd(a, N)$; otherwise, continue;
5 Run the quantum part with inputs a and N $\big($Algorithm 7.2, assume output $\ell_0, ..., \ell_{m-1}\big)$;
6 If $\ell = 0$, go to Step 5;
7 Calculate $b = \ell/q$ (the same q used in the quantum part);
8 Find the convergent of the continued fraction expansion of b with the largest denominator r' such that $r' < N$;
9 If r' is even, calculate $p_1 = \gcd(a^{r'/2} + 1, N)$; otherwise, go to Step 3;
10 If $1 < p_1 < N$, return p_1; otherwise, go to Step 3.

Shor's algorithm is described in Algorithm 7.1 and the quantum part is described in Algorithm 7.2. The circuit is of the quantum part is



where q is the least power of 2 such that $q > N^2$, $m = \log_2 q$, $n = \lceil \log_2 N \rceil$, and a is an integer such that $\gcd(a, N) = 1$. The first register has at least $2n - 1$ qubits (at most $2n$) and the second has exactly n qubits. The states at the bottom of the circuit are used in the analysis of the algorithm. They describe the states of the qubits after each step. Note that state $|\psi_4\rangle$ refers to the first register only. The notation "$/^n$" over a wire denotes that it is a n-qubit register.

Algorithm 7.2: Quantum part of Shor's algorithm

Input: A composite integer N and integer $1 < a < N$ such that $\gcd(a, N) = 1$.

Output: m-bit string ℓ that is the nearest integer to a multiple of q/r with probability greater than $3/\pi^2$, where $q = 2^m$ is the smallest power of 2 such that $q > N^2$.

1 Prepare the initial state $|0\rangle^{\otimes m}|0\rangle^{\otimes n}$, where $m = \lceil 2 \log_2 N \rceil$ and $n = \lceil \log_2 N \rceil$;
2 Apply $H^{\otimes \log_2 q}$ to the first register;
3 Apply $U_N^{(a)}$ to both registers;
4 Measure the second register in the computational basis (assume output $z_0 ... z_{n-1}$);
5 Apply $F_q^\dagger$ to the first register;
6 Measure the first register in the computational basis.

We have presented Shor's algorithm as a Las Vegas algorithm, which means that the output is always correct and the expected runtime is finite. With a small modification, it can be presented as a Monte Carlo algorithm, which means that the output may be incorrect. Then, we focus on the quantum part and we have to show that the probability of returning a nearest integer ℓ to a multiple of q/r is lower bounded by $3/\pi^2$ because this ℓ allows us to obtain r using a continued fraction expansion of ℓ/q.

7.6 Analysis of the quantum part

Calculation of $|\psi_0\rangle$

After the first step, the state of the qubits is

$$|\psi_0\rangle = |0\rangle^{\otimes m}|0\rangle^{\otimes n},$$

where $m = \log_2 q = \lceil 2 \log_2 N \rceil$.

Calculation of $|\psi_1\rangle$

After the second step, the state of the qubits is

$$\begin{aligned} |\psi_1\rangle &= \big(H|0\rangle\big)^{\otimes m} \otimes |0\rangle^{\otimes n} \\ &= \frac{1}{\sqrt{q}} \sum_{\ell=0}^{q-1} |\ell\rangle \otimes |0\rangle^{\otimes n}, \end{aligned}$$

where ℓ is written in the decimal notation and $q = 2^m$.

Calculation of $|\psi_2\rangle$

After the third step, the state of the qubits is

$$\begin{aligned} |\psi_2\rangle &= U_N^{(a)}|\psi_1\rangle \\ &= \frac{1}{\sqrt{q}}\sum_{\ell=0}^{q-1} U_N^{(a)}(|\ell\rangle \otimes |0\cdots 0\rangle). \end{aligned}$$

To simplify $|\psi_2\rangle$, we use the definition of $U_N^{(a)}$ to obtain

$$|\psi_2\rangle = \frac{1}{\sqrt{q}}\sum_{\ell=0}^{q-1} |\ell\rangle \otimes \Big|a^\ell \mod N\Big\rangle,$$

because $(0...0) \oplus a^\ell$ (bitwise XOR) is a^ℓ. From now on, we drop the notation modulo N inside the second ket because we have no XOR operation and there is no danger in failing to recognize the correct arithmetic.

It is really important to understand the structure of $|\psi_2\rangle$ before proceeding. Expanding the sum, we obtain

$$\begin{aligned} \sqrt{q}|\psi_2\rangle = {} & |0\rangle|1\rangle + |1\rangle|a\rangle + |2\rangle|a^2\rangle + \cdots + |r-1\rangle|a^{r-1}\rangle + \\ & |r\rangle|1\rangle + |r+1\rangle|a\rangle + |r+2\rangle|a^2\rangle + \cdots + |2r-1\rangle|a^{r-1}\rangle + \\ & |2r\rangle|1\rangle + |2r+1\rangle|a\rangle + |2r+2\rangle|a^2\rangle + \cdots + |3r-1\rangle|a^{r-1}\rangle + \\ & \vdots \qquad\quad \vdots \qquad\quad \vdots \qquad\quad \vdots \\ & |(c-1)r\rangle|1\rangle + |(c-1)r+1\rangle|a\rangle + |(c-1)r+2\rangle|a^2\rangle + \cdots, \end{aligned}$$

where $c = \lceil q/r \rceil$. Indeed, we have $q = (c-1)r + r_0$, where r_0 is the remainder of q divided by r. The last line has r_0 terms and is incomplete unless $r_0 = 0$.[2] The possible values inside the second ket are 1, a, a^2, ..., a^{r-1}. We have split $|\psi_2\rangle$ into columns that have the same second ket. The first columns have c terms and the last columns have $c-1$ terms.

Calculation of $|\psi_3\rangle$

The fourth step is a measurement of each qubit of the second register, which we assume has returned $z_0, ..., z_{n-1}$. Then there exists r_1 such that $a^{r_1} = z$, where $0 \le r_1 < r$. State $|\psi_2\rangle$ collapses to a superposition of the first register with all terms $|\ell\rangle$ such that $a^\ell \equiv a^{r_1} \mod N$, that is, $\ell = kr + r_1$, for $0 \le k < r$, yielding

$$|\psi_3\rangle = \left(\frac{1}{\sqrt{c}}\sum_{k=0}^{c-1} |kr + r_1\rangle\right)|a^{r_1}\rangle,$$

where $c = \lceil q/r \rceil$ if a^{r_1} belongs to one of the first columns ($r_1 \le r_0$), and $c = \lfloor q/r \rfloor$ if a^{r_1} belongs to one of the last columns ($r_1 > r_0$). The analysis of the algorithm uses parameter c extensively. It is important to memorize its definition and be aware that it is an integer.

[2] The last line is complete if $r \mid q$. In this case, r is a power of 2.

Note that we have renormalized state $|\psi_3\rangle$ as demanded by the measurement postulate. We don't know r_1 because it is selected at random from 0 to $r-1$. The information we wish to acquire, r, is hidden because $kr+r_1$ is a random value from 0 to $q-1$. To perform a measurement of the first register at this point is useless. But, the probability distribution of the first register is a r-periodic function. This motivates the application of the inverse Fourier transform because the result is another (almost) periodic function, whose period is close to q/r.

Calculation of $|\psi_4\rangle$

In the fifth step, we only consider the first register. After applying $F_q^\dagger$ to the state of the first register,[3] we obtain

$$\begin{aligned} |\psi_4\rangle &= \frac{1}{\sqrt{c}} \sum_{k=0}^{c-1} F_q^\dagger |kr + r_1\rangle \\ &= \frac{1}{\sqrt{c\,q}} \sum_{k=0}^{c-1} \sum_{\ell=0}^{q-1} \omega^{-\ell(kr+r_1)} |\ell\rangle. \end{aligned}$$

Rearranging the order of the sums, we obtain

$$|\psi_4\rangle = \frac{1}{\sqrt{q}} \sum_{\ell=0}^{q-1} \omega^{-\ell r_1} \left(\frac{1}{\sqrt{c}} \sum_{k=0}^{c-1} \omega^{-\ell k r} \right) |\ell\rangle.$$

To calculate the sum inside the parentheses, we use again the closed-form formula for the geometric series, which is

$$\sum_{k=0}^{c-1} s^k = \frac{1-s^c}{1-s},$$

if $s \neq 1$. When $s = 1$, the left-hand sum is equal to c. For the sum inside the parentheses, $s = \omega^{-\ell r} = \mathrm{e}^{-2\pi \mathrm{i} \ell r/q}$. Then

$$\sum_{k=0}^{c-1} \omega^{-\ell k r} = \begin{cases} c, & \text{if } q \mid (\ell r), \\ \frac{1-\omega^{-\ell c r}}{1-\omega^{-\ell r}}, & \text{otherwise.} \end{cases}$$

State $|\psi_4\rangle$ simplifies to

$$|\psi_4\rangle = \frac{\sqrt{c}}{\sqrt{q}} \sum_{\substack{\ell=0 \\ q \mid (\ell r)}}^{q-1} \omega^{-\ell r_1} |\ell\rangle + \frac{1}{\sqrt{qc}} \sum_{\substack{\ell=0 \\ q \nmid (\ell r)}}^{q-1} \omega^{-\ell r_1} \frac{1-\omega^{-\ell c r}}{1-\omega^{-\ell r}} |\ell\rangle, \tag{7.1}$$

where the notation $q \nmid (\ell r)$ means that q does not divide ℓr. The first sum is over ℓ such that q is a divisor of (ℓr). The second sum is over the remaining ℓ's.

[3] The algorithm uses $F_q^\dagger$ and not F_q at this point because the goal is to convert a superposition state into a state of the computational basis, not the other way around.

Calculation of the output

The probability of obtaining $0 \le \ell < q$ is

$$p(\ell) \quad = \quad \begin{cases} \frac{c}{q}, & \text{if } q \mid (\ell r), \\ \frac{1}{q\,c}\left|\frac{1-\omega^{-\ell c r}}{1-\omega^{-\ell r}}\right|^2, & \text{otherwise.} \end{cases}$$

Using the definition of ω and $\left|1-\mathrm{e}^{2\pi\mathrm{i}\theta}\right|^2 = 4\sin^2(\pi\theta)$, $p(\ell)$ simplifies to

$$p(\ell) \quad = \quad \begin{cases} \frac{c}{q}, & \text{if } q \mid (\ell r), \\ \frac{\sin^2\frac{\pi\ell r c}{q}}{q\,c\,\sin^2\frac{\pi\ell r}{q}}, & \text{otherwise.} \end{cases} \tag{7.2}$$

The terms that contribute most to $p(\ell)$ when $q \nmid (\ell r)$ are the ones with ℓ satisfying

$$\sin^2\frac{\pi\ell r}{q} \approx 0$$

because $p(\ell)$ is large when the denominator is close to zero. This implies that $\pi\ell r/q$ must be close to a multiple of π (let's say $k\pi$) and then the ℓ's that contribute most are

$$\ell \approx \frac{kq}{r},$$

where k runs from 0 to $r-1$. Note that if ℓ is zero or an exact multiple of q/r then $p(\ell) = c/q \approx 1/r$ — see the definition of $p(\ell)$ in Eq. (7.2). This analysis explains why the peaks of Fig. 7.1 correspond to ℓ's that are close to kq/r. The output of the quantum part is a m-bit string ℓ such that ℓ is close to a multiple of q/r and $m = \log_2 q$.

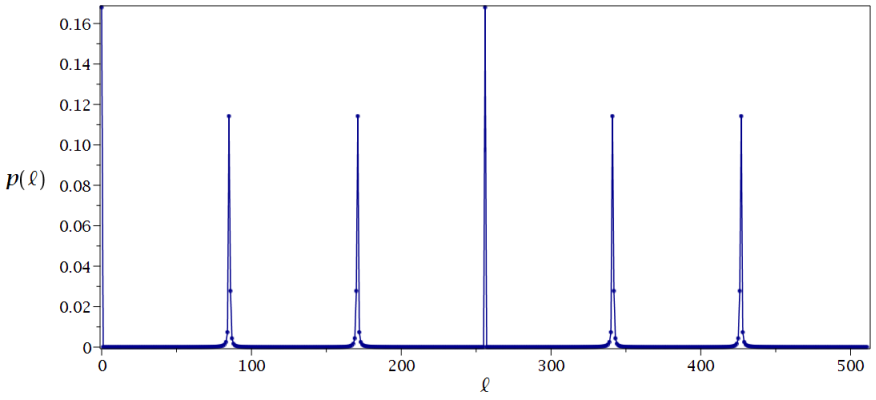


Figure 7.1: Probability distribution $p(\ell)$ as a function of ℓ given by Eq. (7.2) when $N = 21$ ($q = 2^9$, $a = 2$, $r = 6$, $c = 85$). The dots at the top the peaks correspond to $\ell = 0$, 85, 171, 256, 341, 427.

Details about the probability distribution

In the previous analysis, it is missing to show that the numerator of $p(\ell)$ when $q \nmid (\ell r)$, the term

$$\sin^2 \frac{\pi \ell r c}{q},$$

is not too small when $\ell \approx kq/r$ for $0 \leq k < r-1$. Since this analysis is too long when ℓ is a discrete variable, we take an alternative route.

Let us look at $p(\ell)$ as a continuous function in terms of ℓ in the domain $[0, q]$. By using trigonometric identities and the fact that c is an integer, it is straightforward to show that

$$p\left(\ell + \frac{q}{r}\right) = p(\ell).$$

When we look at $p(\ell)$ as a continuous function, we are able to show that it is a truly periodic function, while the function depicted in Fig. 7.1 is not.

Now let us obtain the shape of $p(\ell)$. Since $p(\ell)$ is (q/r)-periodic, let us consider the interval $\ell \in [0, q/r]$, and besides let us restrict to ℓ's such that $q \nmid (\ell r)$. We use the expression

$$p(\ell) \;=\; \frac{\sin^2 \frac{\pi \ell r c}{q}}{q\, c \sin^2 \frac{\pi \ell r}{q}}. \tag{7.3}$$

The numerator of $p(\ell)$, $\sin^2(\pi \ell r c/q)$, is the square of a sinusoidal function, which is (q/rc)-periodic. Note that q/rc is exactly 1 if $r \mid q$ and is close to 1 if $r \nmid q$ because c is either $\lceil q/r \rceil$ or $\lfloor q/r \rfloor$. An example of the numerator of $p(\ell)$ is depicted in Fig. 7.2(a) for $\ell \in [0, q/r]$.

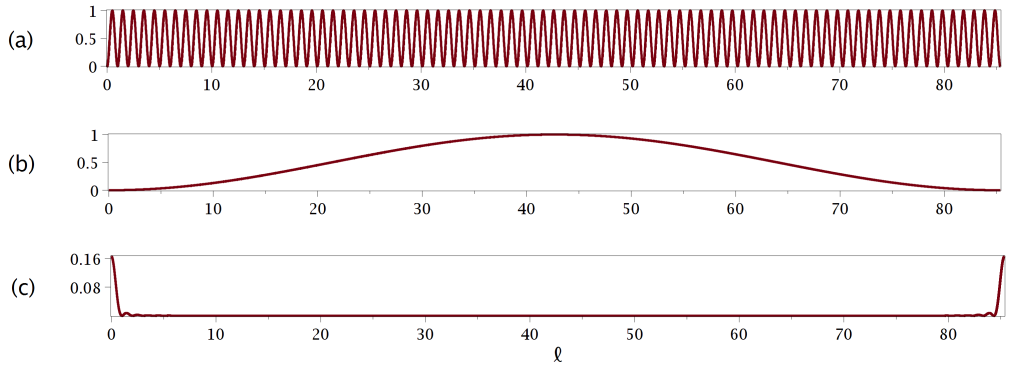


Figure 7.2: (a) Numerator of $p(\ell)$ as a continuous function of ℓ for $\ell \in [0, q/r]$ when $N = 21$ ($q = 2^9$, $r = 6$, $c = 85$). (b) Denominator of $p(\ell)$ as a continuous function of ℓ without qc. (c) $p(\ell)$ as a continuous function of ℓ for $\ell \in [0, q/r]$. Plot (c) is obtained by dividing (a) by (b) by qc.

The denominator of $p(\ell)$ (without qc), $\sin^2(\pi \ell r/q)$, is also the square of a sinusoidal function, which is zero only at $\ell = 0$ and $\ell = q/r$. An example of the denominator of $p(\ell)$ (without qc) for the same values of N, q, and r is depicted in Fig. 7.2(b) for $\ell \in [0, q/r]$.

If we divide the plot shown in Fig. 7.2(a) by the plot shown in Fig. 7.2(b) and divide the result by qc, we obtain the plot shown in Fig. 7.2(c), which is the continuous version of the plot shown in Fig. 7.1 for $0 \leq \ell < q/r$. It is straightforward to check two facts: (1) the largest values

of $p(\ell)$ are close to $\ell = 0$ and $\ell = q/r$ because the denominator of $p(\ell)$ is close to zero; and (2) the smallest values of $p(\ell)$ are in the middle (ℓ close to $q/2r$) because the denominator of $p(\ell)$ is close to qc, which is large—far from zero. The numerator of $p(\ell)$ is oscillating quickly between 0 and 1; this seems confusing, but in reality it isn't. Many more facts about $p(\ell)$ are obtained from this analysis, for instance, it is easy to find the number of zeros of $p(\ell)$. Using our knowledge of calculus, we check that

$$\lim_{\ell=0^+} \frac{\sin^2 \frac{\pi \ell r c}{q}}{\sin^2 \frac{\pi \ell r}{q}} = \lim_{\ell=\frac{q}{r}^-} \frac{\sin^2 \frac{\pi \ell r c}{q}}{\sin^2 \frac{\pi \ell r}{q}} = c^2.$$

Then, $p(0) = p(q/r) = c/q \approx 1/r$. With this analysis, we have obtained the shape of $p(\ell)$ in the whole domain because $p(\ell)$ is (q/r)-periodic, that is, if we put 6 plots of Fig. 7.2(c) side by side, we obtain the continuous version of the plot of Fig. 7.1.

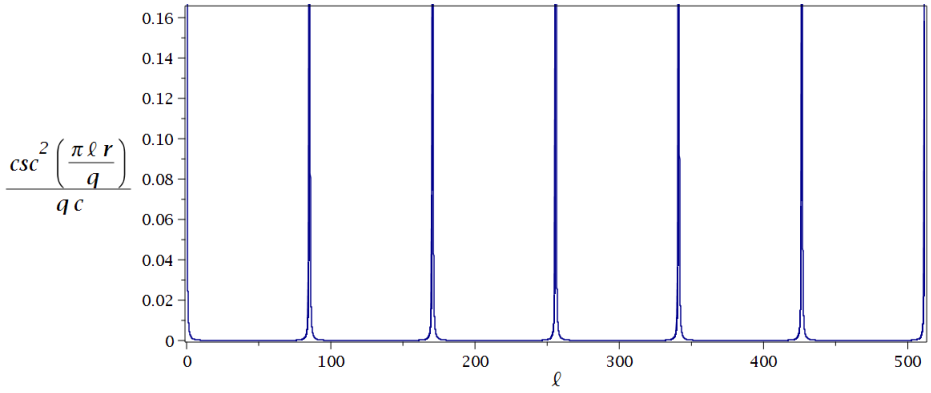


Figure 7.3: The upper part of the envelope of $p(\ell)$ as a function of ℓ in the domain $[0, q]$ when $N = 21$ ($q = 2^9$, $r = 6$, $c = 85$), which is obtained using the square of the cosecant function ($\csc x = 1/\sin x$).

There is an alternative route to obtain the shape of $p(\ell)$ in the domain $[0, q]$. The numerator of $p(\ell)$, $\sin^2 \frac{\pi \ell r c}{q}$, is the square of a sinusoidal function quickly oscillating inside an envelope, the lower part of which is the ℓ-axis and the upper part is the function

$$\frac{\csc^2 \left(\frac{\pi \ell r}{q}\right)}{qc},$$

which is obtained from Eq. (7.3) by replacing the numerator of $p(\ell)$ with 1. Fig. 7.3 depicts the upper part of the envelope in the domain $[0, q]$, which must be compared with Fig. 7.1. Note that the envelope has vertical asymptotes for each ℓ multiple of q/r.

Let us forget about the envelope now and focus on the shape of the first peak of $p(\ell)$. Fig. 7.4 depicts the first peak in detail when $N = 21$, $q = 2^9$, and $r = 6$ showing not only the continuous version in red but also the discrete values in blue when ℓ is an integer. The points in blue are the same ones as depicted in the first peak of Fig. 7.1, and at least four of them are clearly recognizable. Note that the first peak of Fig. 7.1 does not reach c/q because the peak is aborted before reaching the underlying summit. The underlying summit is reached only for ℓ's such that $q \mid (\ell r)$.

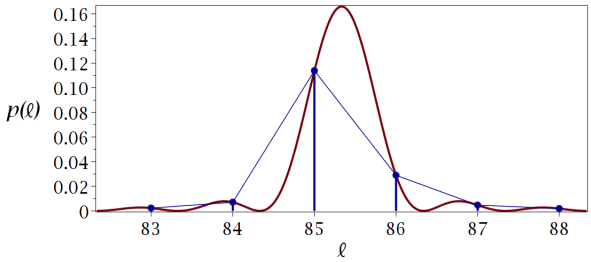


Figure 7.4: First peak of the probability distribution $p(\ell)$ as a function of ℓ when $N = 21$ ($q = 2^9$, $r = 6$, $c = 85$). The red curve is the continuous version of $p(\ell)$ and the height of the peak is $1/r$. The values of $p(\ell)$ for integer ℓ's are highlighted in blue. The blue plot is the stretched version of the first peak of Fig. 7.1.

The particular case $r \mid q$ (r is a power of 2) is remarkable. In this case, c is exactly equal to q/r, which is an integer, and $p(\ell)$ reaches the summit, whose height is exactly $1/r$, for all ℓ's that are multiple of q/r and $p(\ell)$ goes to the bottom of the valley ($p(\ell) = 0$) for all other integer values of ℓ. In Eq. (7.1), $|\psi_4\rangle$ has only the first sum because all terms of the second sum vanish since $\omega^{-\ell cr} = \omega^{-\ell q} = 1$. In this case, the discrete version of the probability distribution is truly periodic with period q/r. We do not need to use the continued fraction expansion to find r. If the result of the measurement is ℓ, we simply take denominator of the reduced fraction of ℓ/q as the candidate for r. This particular case happens because we are using qubits and then the dimension of the Hilbert space is a power of 2.

Success probability of the quantum part

The gist of the analysis is the calculation of the success probability after one round of the quantum part of the algorithm. We have already amassed enough information about the probability distribution; we are ready for the calculation. The trigonometric inequalities

$$4\alpha^2 \leq \sin^2(\pi\alpha) \leq \pi^2\alpha^2 \;\text{ for }\; |\alpha| \leq \frac{1}{2},$$

can be proved using basic calculus. Using both sides of that inequality, we prove the following proposition:

Proposition 7.1. If $|\alpha| \leq 1/2c$ and $c \geq 1$, then

$$\frac{\sin^2 \pi\alpha c}{\sin^2 \pi\alpha} \geq \frac{4c^2}{\pi^2}.$$

Using this proposition, we are able to find an interesting lower bound on the success probability. Let us start by calculating a lower bound on the probability $p(\ell)$ when ℓ is the nearest integer to a multiple of q/r. Suppose that $q \nmid (\ell r)$ and take $\ell = \lfloor kq/r \rceil$ for $1 \leq k < r$, where $\lfloor\,\rceil$ is the notation for the nearest integer. Then,

$$p\left(\left\lfloor \frac{kq}{r} \right\rceil\right) = \frac{\sin^2\left(\pi \left\lfloor \frac{kq}{r} \right\rceil \frac{rc}{q}\right)}{q\,c\,\sin^2\left(\pi \left\lfloor \frac{kq}{r} \right\rceil \frac{r}{q}\right)}.$$

Now we use the trigonometric identity $\sin^2\alpha = \sin^2(\pi k' - \alpha)$ valid for any integer k' to obtain

$$p\left(\left\lfloor \frac{kq}{r} \right\rceil\right) = \frac{\sin^2 \pi\alpha c}{q\, c \sin^2 \pi\alpha},$$

where

$$\alpha = k - \left\lfloor \frac{kq}{r} \right\rceil \frac{r}{q}.$$

Using that

$$\left| \frac{kq}{r} - \left\lfloor \frac{kq}{r} \right\rceil \right| \leq \frac{1}{2}, \tag{7.4}$$

we obtain

$$|\alpha| \leq \frac{r}{2q} \leq \frac{1}{2\left\lfloor \frac{q}{r} \right\rfloor} = \frac{1}{2c},$$

where $c = \lfloor q/r \rfloor$. Using Proposition 7.1, we obtain

$$p\left(\left\lfloor \frac{kq}{r} \right\rceil\right) \geq \frac{4c}{\pi^2 q}.$$

Using $c \geq q/r - 1$ and $q > N^2 > rN$, we obtain

$$p\left(\left\lfloor \frac{kq}{r} \right\rceil\right) > \frac{4}{\pi^2 r}\left(1 - \frac{1}{N}\right). \tag{7.5}$$

We can visualize this result by noticing that there are two blue bars inside the peak of $p(\ell)$ in Fig. 7.4. Since $N > 4$, we have just shown that the height of the largest bar is larger than $3/(\pi^2 r)$, that is, it is never smaller that 30% of the height of the peak.

In the quantum part of the algorithm, there are r possible k's. Then, a lower bound on the success probability is

$$p_{\text{succ}} > \frac{3}{\pi^2},$$

which means that the quantum part has a chance of at least 30% of returning a m-bit string ℓ that is the nearest integer to a multiple of q/r. Numerical calculations with N up to 4 figures show that the lower bound can be improved to at least 70%. Note that this success probability includes the worthless result $\ell = 0$, which is a multiple of q/r.

7.7 Analysis of the classical part

Before running the quantum part of the algorithm, we need to do a classical checklist. It is necessary to check whether N is a composite number, which can be done efficiently using primality-testing algorithms[4]. We quickly check whether N is even. Besides, there are efficient classical algorithms to check whether N is a power of some prime number p [7]. After assuring ourselves that N is a composite odd number and is not a power of a prime number, we select a random integer a such that $1 < a < N$ and check whether $\gcd(a, N) = 1$, which can be done efficiently using the Euclidean algorithm.

[4] https://en.wikipedia.org/wiki/Primality_test

Then, after running the quantum part of the algorithm, we assume that the output is a $(\log_2 q)$-bit string ℓ such that ℓ is the nearest integer to a multiple of q/r, that is

$$\ell = \left\lfloor \frac{kq}{r} \right\rceil$$

for some k such that $0 \leq k < r$. If $\ell = 0$, which happens with a negligible probability when N is not too small, we have to rerun the quantum part of the algorithm. Define

$$b = \frac{\ell}{q},$$

which obeys $0 < b < 1$. Now we use the method of continued fraction approximation to obtain the desired information r. A continued fraction expansion of a positive rational number $b < 1$ is

$$b = \cfrac{1}{b_1 + \cfrac{1}{b_2 + \cfrac{1}{\ddots + \frac{1}{b_z}}}},$$

where b_1 to b_z are positive integers, and z is a natural number. The notation for the continued fraction is $[b_1, b_2, ..., b_z]$, and the successive convergents are $[b_1]$, $[b_1, b_2]$, $[b_1, b_2, b_3]$, and so on, each one closer to b, until the last one, which is equal to b. Each convergent is converted into a rational number by truncating the continued fraction expansion. In the algorithm, we have to find the convergent $[b_1, b_2, ..., b_j]$ that has the largest j such that the denominator of the equivalent rational number is less than N. The denominator of this convergent is the candidate for r.

For example, the successive convergents of ℓ/q for $\ell = 85$ and $q = 2^9$ are $[6] = 1/6$, $[6, 42] = 42/253$, and $[6, 42, 2] = 85/512 = \ell/q$. When $N = 21$ and $a = 2$, we have to select the convergent $[6]$ and then the candidate for r is the denominator of $1/6$. Luckily, $a^6 \equiv 1 \mod N$. Our luck would fail if we picked $\ell = 171$ (see Fig. 7.1) because we would conclude that $r = 3$.

Now we can see why we have to demand that $q > N^2$. It is a consequence of a theorem proved at the end of the chapter on continued fraction expansion in Hardy and Wright's book [25].

Theorem 7.2. (Hardy and Wright) If k and r are positive integers and b is a positive real number and

$$\left| \frac{k}{r} - b \right| < \frac{1}{2r^2},$$

then k/r is a convergent of the continued fraction expansion of b.

In Eq. (7.4), we have shown that the output ℓ of the quantum part obeys

$$\left| \frac{k}{r} - \frac{\ell}{q} \right| \leq \frac{1}{2q}, \tag{7.6}$$

for some integer k such that $0 \leq k < r$, which is unknown to us as well as r. In order to use Theorem 7.2, we have to demand that $q > N^2$ because N is an upper bound for r, that is, $1/q < 1/N^2 < 1/r^2$. Then, we can use the theorem and be sure that k/r is a convergent of ℓ/q.

To understand why we have to pick the convergent $[b_1, b_2, ..., b_j]$ that has the largest j such that the denominator of the equivalent rational number is less than N, we need to consider the

following facts. If we pick a convergent $[b_1, b_2, ..., b_{j'}] = k'/r'$ that obeys Eq. (7.6) and $r' < N$ and it does not have the largest j', then the next convergent, let us say $[b_1, b_2, ..., b_{j+1}] = k''/r''$, also obeys Eq. (7.6) and $r'' < N$. We obtain a contradiction because on the one hand using Eq. (7.6) twice we have

$$\left|\frac{k'}{r'} - \frac{k''}{r''}\right| = \left|\left(\frac{k'}{r'} - b\right) - \left(\frac{k''}{r''} - b\right)\right| \leq \frac{1}{q} < \frac{1}{N^2},$$

and on the other hand we have

$$\left|\frac{k'}{r'} - \frac{k''}{r''}\right| = \frac{|k'r'' - k''r'|}{r'r''} > \frac{1}{N^2}.$$

The last inequality follows from the inequalities $|k'r'' - k''r'| \geq 1$ and $r' < N$ and $r'' < N$. In conclusion, there is only one convergent $[b_1, b_2, ..., b_j]$ that obeys Eq. (7.6) such that the denominator of the equivalent fraction is less than N; it is the one with the largest j.

The analysis is not done yet because it may happen that $\gcd(k, r) > 1$. In this case, when we look at the denominator of k/r, we obtain a factor of r, not r itself. In the example above with $\ell = 171$, we have $k = 2$ and $r = 6$ and the denominator of k/r yields a wrong result. We have to discard those cases and we ask how many k's relatively prime with r are there. The proportion of good k's is $\varphi(r)/r$, where $\varphi(r)$ is Euler's totient function. There is a lower bound for $\varphi(r)$ given by [25, 55]

$$\varphi(r) > \frac{r}{4\ln(\ln(r))}, \quad \text{for } r \geq 7.$$

Then, a lower bound for the probability that the output ℓ is the nearest integer to a multiple of q/r and $\gcd(k, r) = 1$ is

$$p\left(\ell = \left\lfloor \frac{kq}{r} \right\rceil \text{ and } \gcd(k, r) = 1\right) > \frac{3}{4\pi^2 \ln(\ln(r))},$$

for $r \geq 7$.

If we think of the factoring algorithm as a Monte Carlo algorithm, which is obtained by removing the "go to" statements of Algorithm 7.1, the algorithm will run only one time and a lower bound on the overall success probability is

$$\frac{9}{16\pi^2 \ln(\ln(r))},$$

which is obtained from the analysis of this Section and Fact 3 $\big(r$ must be even and $\gcd(a^{r/2} + 1, N) > 1\big)$. If we think of the factoring algorithm as a Las Vegas algorithm, the way it is described in Algorithm 7.1, the success probability is 1 but we have to calculate an upper bound for the average number of times the quantum part of the algorithm will run until finding a factor of N. Suppose that an algorithm has probability $0 < p < 1$ of outputting the correct result in one run of the algorithm. The probability of outputting the correct result after exactly n runs is $(1-p)^{n-1}p$ because it must fail $n-1$ times before succeeding. The average number of times

the algorithm will run is[5]

$$\sum_{n=1}^{\infty} n\,(1-p)^{n-1}p \;=\; \frac{1}{p}.$$

Then, an upper bound on the average number of times that the quantum part will run in Algorithm 7.1 is $16\pi^2\ln(\ln(r))/9$ for $r \geq 7$. Now we are done.

7.8 Circuit of the Fourier transform

The first decomposition of the Fourier transform in terms of basic gates is in an IBM report of 1994, which became widely available in 2002 [15]. A description of this decomposition based on the classical FFT is available in [39].

The basic block of the circuit of the Fourier transform F_{2^n}, where n is the number of qubits, is the controlled gate $C(R_k)$ for $k \geq 0$, where

$$R_k \;=\; \begin{bmatrix} 1 & 0 \\ 0 & \exp\left(\frac{2\pi \mathrm{i}}{2^k}\right) \end{bmatrix}.$$

The matrix representation of $C(R_k)$ is

$$C(R_k) = \begin{bmatrix} I_2 & \\ & R_k \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & \exp\left(\frac{2\pi \mathrm{i}}{2^k}\right) \end{bmatrix}.$$

The set of gates R_k has many special subcases, which are

$$\begin{aligned} R_0 &= I_2, \\ R_1 &= Z, \\ R_2 &= S, \\ R_3 &= T, \end{aligned}$$

where Z, S, and T are the Pauli Z gate, phase gate, and $\pi/8$ or T gate, respectively. Note that, $R_{k+1} = \sqrt{R_k}$. Then, the sequence above means the next gate is the square root of the previous one. The same idea applies to $C(R_k)$, that is, $C(R_{k+1}) = \sqrt{C(R_k)}$, but in the latter case we are calculating the square root of (4×4)-matrices.

Fig. 7.5 depicts the circuit of F_{2^n} in terms of the Hadamard, $C(R_k)$, and swap gates when $n = 5$. The structure of the circuit can be easily grasped from this example. The circuit has $n+1$ blocks. From left to right, the first block has n gates, starting with H and then R_2, R_3,..., R_n acting on qubit 1 and controlled by qubit 2, 3,..., n, respectively. The next block starts again with H and then R_2, R_3,..., R_{n-1} acting on qubit 2 and controlled by qubit 3, 4,..., n, respectively. This goes on until we reach the last qubit, on which a single H (n-th block) is

[5] The sum is calculated using that

$$\sum_{n=1}^{\infty} nq^n \;=\; q\,\frac{\mathrm{d}}{\mathrm{d}q}\left(\sum_{n=1}^{\infty} q^n\right),$$

where $q = 1 - p$. The sum inside the derivative is the geometric series, whose value is $q/(1-q)$ when $|q| < 1$.

```
In [1]: from sympy.physics.quantum.qft import QFT
        from sympy.physics.quantum.circuitplot import circuit_plot

In [2]: n=5
        c = circuit_plot(QFT(0,n).decompose(), nqubits=n)
```

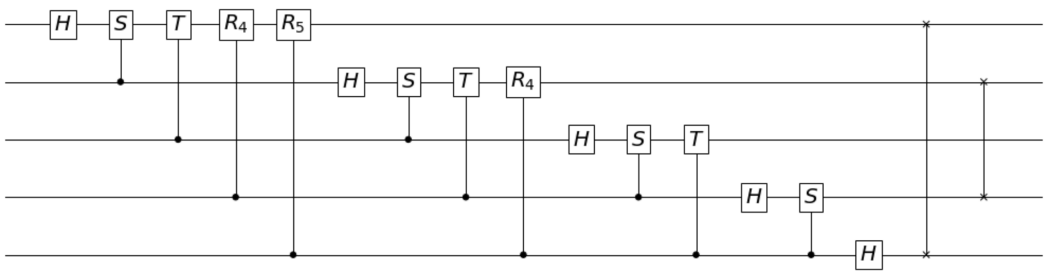


Figure 7.5: Snapshot of a Jupyter notebook showing the decomposition of the Fourier transform F_{2^5} using Python commands *QFT* and *circuit_plot* of the Sympy library.

applied. The last block is made of $\lfloor n/2 \rfloor$ swap gates and has a simple symmetric structure. If n is odd, the central qubit does not swap. The number of gates is

$$n + (n-1) + \cdots + 1 + \left\lfloor \frac{n}{2} \right\rfloor \;=\; \frac{n(n+1)}{2} + \left\lfloor \frac{n}{2} \right\rfloor .$$

Let us show that a circuit with the structure depicted in Fig. 7.5 implements the Fourier transform when we have n qubits. Suppose that the input is $|\ell\rangle = |\ell_1\rangle \otimes \cdots \otimes |\ell_n\rangle$. When the input is a state of the computational basis, the output is an unentangled state $|\psi_1\rangle \otimes \cdots \otimes |\psi_n\rangle$. Let us start by calculating the output $|\psi_1\rangle$ of the first qubit. Since there is a swap gate inverting the states of the first and last qubit, we have

$$|\psi_1\rangle = H|\ell_n\rangle = \frac{|0\rangle + \mathrm{e}^{2\pi \mathrm{i} \frac{\ell_n}{2}}|1\rangle}{\sqrt{2}} = \frac{|0\rangle + \mathrm{e}^{2\pi \mathrm{i} \frac{\ell}{2}}|1\rangle}{\sqrt{2}}.$$

The second equation follows from the fact that if ℓ_n is 0, the output is $|+\rangle$, and if $\ell_n = 1$, the output is $|-\rangle$ (because $\mathrm{e}^{\pi \mathrm{i}}$ is -1). The last equation follows from the decomposition $\ell = 2^{n-1}\ell_1 + 2^{n-2}\ell_{n-2} + \cdots + 2\ell_{n-1} + \ell_n$ and from $\mathrm{e}^{2\pi \mathrm{i} k} = 1$ if k is an integer.

The output of the second qubit is

$$|\psi_2\rangle = R_2^{\ell_n} H|\ell_{n-1}\rangle = R_2^{\ell_n} \left(\frac{|0\rangle + \mathrm{e}^{2\pi \mathrm{i} \frac{\ell_{n-1}}{2}}|1\rangle}{\sqrt{2}} \right) = \frac{|0\rangle + \mathrm{e}^{2\pi \mathrm{i} \left(\frac{\ell_{n-1}}{2} + \frac{\ell_n}{2^2} \right)}|1\rangle}{\sqrt{2}} = \frac{|0\rangle + \mathrm{e}^{2\pi \mathrm{i} \frac{\ell}{2^2}}|1\rangle}{\sqrt{2}}.$$

The first equation is obtained from Fig. 7.5 using that $C(R_2)|\ell_n\rangle|\ell_{n-1}\rangle = |\ell_n\rangle\big(R_2^{\ell_n}|\ell_{n-1}\rangle\big)$. The second equation uses the same calculation described before for the first qubit. The third equation follows from

$$R_2^{\ell_n}|0\rangle = |0\rangle,$$
$$R_2^{\ell_n}|1\rangle = \mathrm{e}^{2\pi \mathrm{i} \frac{\ell_n}{2^2}}|1\rangle.$$

The last equation uses the same decomposition of ℓ described before.

The output of the last qubit is

$$|\psi_n\rangle = R_n^{\ell_n}\cdots R_2^{\ell_2}H|\ell_1\rangle = \frac{|0\rangle + \mathrm{e}^{2\pi\mathrm{i}\left(\frac{\ell_1}{2}+\frac{\ell_2}{2^2}+\cdots+\frac{\ell_n}{2^n}\right)}|1\rangle}{\sqrt{2}} = \frac{|0\rangle + \mathrm{e}^{2\pi\mathrm{i}\frac{\ell}{2^n}}|1\rangle}{\sqrt{2}}.$$

The first equation is obtained from Fig. 7.5 using that the output of the last qubit is obtained from the action of H, $R_2^{\ell_2}$, ..., $R_n^{\ell_n}$ on the first qubit. The second and third equations are obtained with the same sort of calculations described before for the first and second qubits.

Then, the output $|\psi_1\rangle\otimes\cdots\otimes|\psi_n\rangle$ of the circuit of Fig. 7.5 with n qubits is

$$\frac{|0\rangle + \mathrm{e}^{2\pi\mathrm{i}\frac{\ell}{2}}|1\rangle}{\sqrt{2}} \otimes \frac{|0\rangle + \mathrm{e}^{2\pi\mathrm{i}\frac{\ell}{2^2}}|1\rangle}{\sqrt{2}} \otimes\cdots\otimes \frac{|0\rangle + \mathrm{e}^{2\pi\mathrm{i}\frac{\ell}{2^n}}|1\rangle}{\sqrt{2}}.$$

Now we convert each term into a sum

$$\frac{1}{\sqrt{2}}\sum_{k_1=0}^{1}\mathrm{e}^{2\pi\mathrm{i}k_1\frac{\ell}{2}}|k_1\rangle \otimes \frac{1}{\sqrt{2}}\sum_{k_2=0}^{1}\mathrm{e}^{2\pi\mathrm{i}k_2\frac{\ell}{2^2}}|k_2\rangle\otimes\cdots\otimes\frac{1}{\sqrt{2}}\sum_{k_n=0}^{1}\mathrm{e}^{2\pi\mathrm{i}k_n\frac{\ell}{2^n}}|k_n\rangle.$$

Pushing all sums to the right-hand side and combining the exponentials, we obtain

$$\frac{1}{\sqrt{2^n}}\sum_{k_1,\ldots,k_n=0}^{1}\mathrm{e}^{2\pi\mathrm{i}\ell\left(\frac{k_1}{2}+\cdots+\frac{k_n}{2^n}\right)}|k_1,\ldots,k_n\rangle,$$

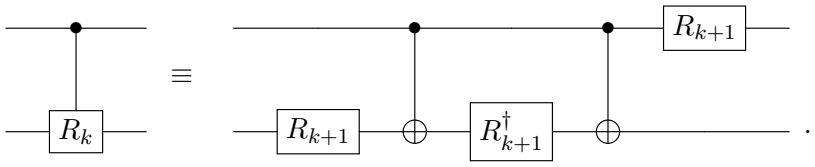
which is equivalent to

$$\frac{1}{\sqrt{2^n}}\sum_{k=0}^{2^n-1}\mathrm{e}^{\frac{2\pi\mathrm{i}\ell k}{2^n}}|k\rangle.$$

Using the definition of the Fourier transform given in Sec. 7.4, we recognize that the last expression is $F_{2^n}|\ell\rangle$.

Decomposition of $C(R_k)$

The circuit that provides the decomposition of $C(R_k)$ in terms of CNOT and 1-qubit gates R_{k+1} is



This is not a decomposition in terms of universal gates because we need to write R_{k+1} in terms of a finite set of 1-qubit gates. In most cases, it is unnecessary to worry about this step, since it is done automatically by the compiler of quantum computers. R_k can be implemented using R_z, which is a rotation of the Bloch sphere about the z-axis. Note that if k is large, errors will prevent these gates to work properly unless error-correcting codes are present.

Now we show that the decomposition of $C(R_k)$ in terms of CNOT and 1-qubit gates R_{k+1} is correct. If the input to $C(R_k)$ is $|j\rangle|\ell\rangle$ then the output is

$$|j\rangle|\ell\rangle \xrightarrow{\bullet\!-\boxed{R_k}} |j\rangle R_k^j|\ell\rangle.$$

If $|j\ell\rangle$ is equal to $|00\rangle$ or $|01\rangle$ or $|10\rangle$, the output is $|j\ell\rangle$ because $R_k|0\rangle = |0\rangle$. The only nontrivial output is obtained when the input is $|11\rangle$. In this case, the output is $\exp(2\pi\mathrm{i}/2^k)|11\rangle$.

Let us analyze the decomposition (right-hand circuit) to check that the result is the same. For the inputs $|00\rangle$ or $|01\rangle$ or $|10\rangle$, either the CNOTs are not activated or they cancel out. Besides, since $R_k|0\rangle = |0\rangle$ and $R_{k+1}^\dagger R_{k+1} = I$, we check that the output is equal to the input. The missing case is input $|11\rangle$. Let us follow each step at a time using that $R_{k+1}|0\rangle = |0\rangle$ and $R_{k+1}|1\rangle = \exp(2\pi\mathrm{i}/2^{k+1})|1\rangle$:

$$|1\rangle|1\rangle \xrightarrow{I\otimes R_{k+1}} \mathrm{e}^{2\pi\mathrm{i}/2^{k+1}}|1\rangle|1\rangle \xrightarrow{\bullet-\oplus} \mathrm{e}^{2\pi\mathrm{i}/2^{k+1}}|1\rangle|0\rangle \xrightarrow{I\otimes R_{k+1}^\dagger} \mathrm{e}^{2\pi\mathrm{i}/2^{k+1}}|1\rangle|0\rangle \to \text{ - - -}$$

$$\text{- - -} \xrightarrow{\bullet-\oplus} \mathrm{e}^{2\pi\mathrm{i}/2^{k+1}}|1\rangle|1\rangle \xrightarrow{R_{k+1}\otimes I} \mathrm{e}^{2\pi\mathrm{i}/2^{k+1}}\mathrm{e}^{2\pi\mathrm{i}/2^{k+1}}|1\rangle|1\rangle = \mathrm{e}^{2\pi\mathrm{i}/2^k}|1\rangle|1\rangle.$$

Note that the output is $\exp(2\pi\mathrm{i}/2^k)|11\rangle$, which is equal to the output of the left-hand circuit.

7.9 Final remarks

In his original paper [59], Shor showed that the average number of times we have to rerun the algorithm until hitting a solution is $O(\log\log r)$ and the asymptotic lower bound on the success probability is $4/\pi^2$, which is obtained from Eq. (7.5) when N is large. It is possible to reduce the number of repetitions of the quantum part by improving the classical post-processing [22].

Chapter 8

Shor's Algorithm for Discrete Logarithm

The full paper on Shor's algorithms was published in 1997 [58] and has described not only an algorithm for integer factoring but also an exponentially faster algorithm for discrete logarithm. It is a remarkable and celebrated scientific contribution to quantum computing, but the algorithm for discrete logarithm has not been described in many books [30]. Some papers apply this algorithm to cryptography [29, 51].

8.1 Preliminaries on number theory

The *discrete logarithm* of b to base a, denoted by $\log_a b$, is the number of times that a multiplied by itself is b, that is, $a^s = b$. Here we are assuming that a, b, and s are positive integer numbers. For instance, $\log_2 8$ is 3 because $2 \times 2 \times 2 = 8$. Since $s = \log_a b$, we write

$$a^{\log_a b} = b.$$

Note that there are choices of a and b so that the discrete logarithm $\log_a b$ does not exist, for instance, $\log_2 7$ because there is no integer s that satisfies $2^s = 7$. Two important properties of the logarithm to base a, when $\log_a b$ and $\log_a c$ exist, are

- $\log_a(bc) = \log_a b + \log_a c$;
- $\log_a(b^c) = c \log_a b$.

There are two drawbacks in the context of quantum algorithms when we use the set of positive integers $\mathbb{Z}^+$ to define the notion of discrete logarithm: (1) Depending on the choice of a and b, $\log_a b$ may not exist, and (2) if $\log_a b$ do exist, there are efficient methods to calculate $\log_a b$ using the fact that the discrete logarithm is equal to the standard definition of *logarithm* of real numbers.

Instead of using $\mathbb{Z}^+$, let us use a subset of the set $\mathbb{Z}_N = \{0, 1, ..., N-1\}$, which has two modular operations—addition and multiplication. We are basically interested in the multiplication modulo N, where $N > 1$. To pick up the desired subset of $\mathbb{Z}_N$, we need to know which numbers of $\mathbb{Z}_N$ have inverse. A number $a \in \mathbb{Z}_N$ has an inverse a^{-1} ($aa^{-1} \equiv 1 \mod N$) if and only if the greatest common divisor of a and N is 1. If a has an inverse, the set $G_a = \{a, a^2, ..., a^r\}$ is a *cyclic*

group, where r is the order of a modulo N (order(a) is the smallest positive integer so that $a^r \equiv 1 \mod N$). The group's product in this case is the multiplication modulo N. If we select any element b in G_a, $\log_a b$ exists. For example, for $N = 34$ we have $G_{15} = \{15, 21, 9, 33, 19, 13, 25, 1\}$, and $\log_{15} 15 = 1$, $\log_{15} 21 = 2$, $\log_{15} 9 = 3$, $\log_{15} 33 = 4$, and so on. In this context, nobody knows a classical algorithm that calculates $\log_a b$ efficiently (in polynomial time) for an arbitrary b in G_a.

Shor's factoring algorithm can calculate $\log_a b$ efficiently if order(b) is a nontrivial factor of order(a) because $\log_a b$ =order(a)/order(b) in this case. Before trying to calculate $\log_a b$ using discrete-logarithm algorithms, we check whether the order of b divides the order of a. If not, then we use a quantum algorithm that calculates $\log_a b$ efficiently for an arbitrary b in G_a.

The main strategy in the part of Shor's algorithm that calculates the order of $a \in \mathbb{Z}_N$ is to define the function

$$\begin{aligned} f\colon \mathbb{Z}_{2^m} &\to \mathbb{Z}_N \\ x &\mapsto a^x, \end{aligned}$$

where $m \approx 2n$, $n = \lceil \log_2 N \rceil$, and to exploit the fact that f is r-periodic. We know that this function can be implemented in a quantum computer with around $3n$ qubits using the unitary operator U_f defined by $U_f|x\rangle|y\rangle = |x\rangle|y \oplus (a^x \mod N)\rangle$. It is possible to determine r efficiently with one application of the inverse discrete Fourier transform $F_{2^m}^\dagger$ to the first register after U_f has been applied to a superposition of all states of the computational basis of the first register (the second register is initially set to $|0\rangle$). The method works because f has period r.

It is natural to ask whether the same strategy can be used to calculate s knowing that $a^s \equiv b \mod N$. The problem is that there is no s-periodic function $f(x)$ with domain $\mathbb{Z}_{2^m}$ and codomain $\mathbb{Z}_N$ that uses parameters a and b only. The way out is to use a function $f(x, y)$ with two variables defined as

$$\begin{aligned} f\colon \mathbb{Z}_r \times \mathbb{Z}_r &\to \mathbb{Z}_N \\ (x, y) &\mapsto a^x b^y. \end{aligned}$$

This definition has two main differences: (1) The domain of each variable is $\mathbb{Z}_r$ instead of $\mathbb{Z}_{2^m}$, and (2) f has two variables. Regarding (1), it is important to use the domain $\mathbb{Z}_r$; otherwise, the modular arithmetic would fail to provide the correct answer at the end of the algorithm. This means that r must be calculated beforehand and if r is not a power of 2, the implementation of the Fourier transform F_r requires extra labor. Regarding (2), f is periodic in the following way:

$$f(x, y) = f(x + kr - \ell s, y + \ell),$$

where $k, \ell \in \mathbb{Z}_r$. In fact,

$$a^x b^y \equiv a^x (b)^{-\ell} b^y b^\ell \equiv a^x a^{kr} (a^s)^{-\ell} b^y b^\ell \equiv a^{x+kr-\ell s} b^{y+\ell} \mod N,$$

where we have used that $a^r \equiv 1$ and $a^s \equiv b \mod N$. To better understand the periodicity of $f(x, y)$, we have to use the concept of lattices and sub-lattices in finite vector spaces.

8.2 Lattices in finite vector spaces

There are many kinds of lattices in mathematics. The ones we are interested in are the lattices in finite vector spaces, in particular, two-dimensional lattices that are additive subgroups of the

vector space $\mathbb{Z}_r \times \mathbb{Z}_r$ (denoted by $\mathbb{Z}_r^2$ from now on). This vector space is the set of vectors with two entries each one in $\mathbb{Z}_r$, whose canonical basis is $\{\mathbf{e}_0, \mathbf{e}_1\}$, where

$$\mathbf{e}_0 = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \text{ and } \mathbf{e}_1 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

This means that any vector $\mathbf{v}$ in $\mathbb{Z}_r^2$ is a linear combination of $\mathbf{e}_0$ and $\mathbf{e}_1$, that is, there are scalars $k, \ell \in \mathbb{Z}_r$ so that $\mathbf{v} = k\mathbf{e}_0 + \ell\mathbf{e}_1$.

A lattice L in $\mathbb{Z}_r^2$ is defined as a set

$$L = \{k\mathbf{r} + \ell\mathbf{s} : k, \ell \in \mathbb{Z}_r\},$$

where $\mathbf{r}$ and $\mathbf{s}$ are two linearly independent vectors in $\mathbb{Z}_r^2$. Set $\{\mathbf{r}, \mathbf{s}\}$ is a basis of L. Different bases can span the same lattice. Note that $\mathbb{Z}_r^2$ itself is a lattice in $\mathbb{Z}_r^2$. Then, L is a sub-lattice of $\mathbb{Z}_r^2$.

Vectors can be represented by points. For instance, vectors

$$\mathbf{r} = \begin{pmatrix} 16 \\ 0 \end{pmatrix} \text{ and } \mathbf{s} = \begin{pmatrix} 5 \\ 1 \end{pmatrix}$$

are shown in Fig. 8.1 as points $(x, y) = (16, 0)$ and $(5, 1)$. The figure also shows the lattice with basis $\{(16, 0), (5, 1)\}$ in $\mathbb{Z}_{16}^2$.

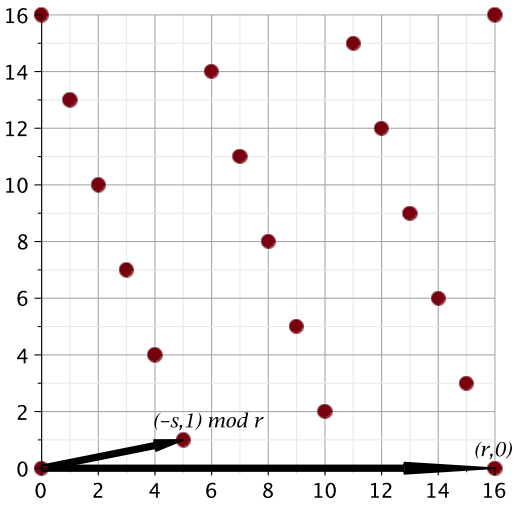


Figure 8.1: Lattice with basis $\mathbf{r} = (16, 0)$ and $\mathbf{s} = (-11, 1)$ modulo 16 and cyclic boundaries.

As an example of application to the calculation of discrete logarithms, consider again $N = 34$ but this time take $a = 27$ and

$$G_{27} = \{27, 15, 31, 21, 23, 9, 5, 33, 7, 19, 3, 13, 11, 25, 29, 1\}.$$

Suppose we want to calculate $\log_{27} 3$, whose answer we know: $s = 11$. Since the order of $a = 27$ is $r = 16$ modulo 34, the lattice L with basis $\mathbf{r} = (r, 0)$ and $\mathbf{s} = (-s, 1)$ shown in Fig. 8.1 represents all points (x, y) in $\mathbb{Z}_{16}^2$ so that $f(x, y) = f(0, 0)$, where $f(x, y) = a^x b^y \mod 34$, $a = 27$, and

$b = 3$. In fact, $f(k\mathbf{r} + \ell\mathbf{s}) = f(kr - \ell s, \ell) = f(0, 0)$. So, f is a periodic function where the period depends on both s and r. The lattice L is a subgroup of the additive group $\mathbb{Z}_{16}^2$. This means that we can partition $\mathbb{Z}_{16}^2$ into equivalence classes, the cosets of L in $\mathbb{Z}_{16}^2$, and we may select the following representatives: $(0, 0)$, $(1, 0)$, ..., $(r - 1, 0)$. The image of f on any point $(x, y) \in \mathbb{Z}_r^2$ is equal to the image f on one of the representatives. That is, $f(x - \ell s, \ell) = f(x, 0)$ for $0 \leq \ell < r$, which corresponds to shifting the lattice x unities to the right with cyclic boundary conditions, that is, to add $(x, 0)$ to each element of L modulo 16. When we fix x and run ℓ from 0 to $r - 1$ we cover the coset with representative $(x, 0)$.

8.3 Special case: The order of a is a power of 2

Let N, a, and b be known positive integers and let s be a positive integer such that $a^s \equiv b \mod N$ and $\gcd(a, N) = 1$. Our goal is to find s given N, a, and b as input. Let r be the order of a modulo N, which can be efficiently determined using Shor's factoring algorithm. We address in this Section the case $r = 2^m$ for some integer m, which means that r is a power of 2. In this case, there is an efficient classical algorithm called *Pohlig–Hellman algorithm* that is able to calculate s in polynomial time. We describe the quantum algorithm for this case because the Fourier transform F_r can be implemented in a straightforward way in a qubit-based quantum computer and the analysis of the algorithm is easier than the general case.

Let f be a two-variable function with domain $\mathbb{Z}_r \times \mathbb{Z}_r$ and codomain $\mathbb{Z}_N$ defined as

$$f(x, y) = a^x b^y \mod N.$$

We have shown that f is periodic in the following way:

$$f(x, y) = f(x + kr - \ell s, y + \ell).$$

Using f, we define a 3-register unitary operator

$$U_f|x\rangle|y\rangle|z\rangle = |x\rangle|y\rangle|z \oplus f(x, y)\rangle, \tag{8.1}$$

where the first and second registers have m qubits each and the third register has $n = \lceil \log_2 N \rceil$ qubits. The arithmetic with the variables of the first and second registers is performed modulo r. The arithmetic to calculate the image $f(x, y)$ is performed modulo N. The symbol $\oplus$ represents the bitwise xor operation. The algorithm that calculates the discrete logarithm when r is a power of 2 is described in Algorithm 8.1 and the circuit is depicted in Fig. 8.2.

Analysis of the algorithm

In step 2, we apply $H^{\otimes m} \otimes H^{\otimes m} \otimes I$ to $|0\rangle|0\rangle|0\rangle$ obtaining

$$|\psi_1\rangle = \frac{1}{r} \sum_{x,y=0}^{r-1} |x\rangle|y\rangle|0\rangle.$$

In step 3, we apply U_f to $|\psi_1\rangle$ obtaining

$$|\psi_2\rangle = \frac{1}{r} \sum_{x,y=0}^{r-1} |x\rangle|y\rangle|f(x, y)\rangle.$$

Algorithm 8.1: Discrete logarithm algorithm when r is a power of 2

Input: N, a, b, and r (order of a).

Output: $s = \log_a b$ with probability 1/2, where $a^s \equiv b \mod N$

1 Prepare the initial state $|0\rangle^{\otimes m}|0\rangle^{\otimes m}|0\rangle^{\otimes n}$, where $m = \log_2 r$ and $n = \lceil \log_2 N \rceil$;

2 Apply $H^{\otimes m} \otimes H^{\otimes m}$ to the first and second registers;

3 Apply U_f as defined in Eq. (8.1);

4 Measure the third register in the computational basis;

5 Apply $F_r^\dagger \otimes F_r^\dagger$ to the first and second registers;

6 Measure the first and second registers in the computational basis, where r_1 and r_2 are the results;

7 If $\gcd(r_1, r) = 1$, return $s \equiv r_2/r_1 \mod r$; otherwise, Fail.

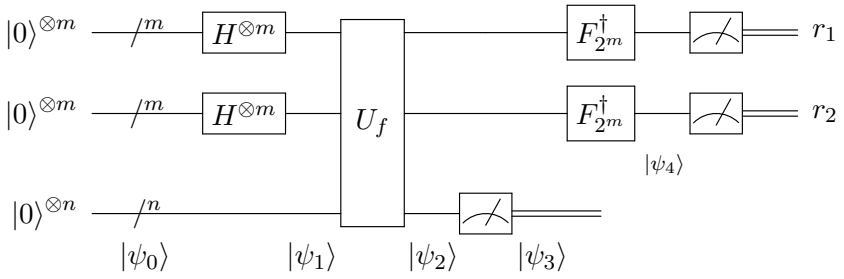


Figure 8.2: Circuit of the discrete logarithm algorithm when r is a power of 2, where $m = \log_2 r$, $n = \lceil \log_2 N \rceil$, and U_f is defined by Eq. (8.1). If $\gcd(r_1, r) = 1$, return $s \equiv r_2/r_1 \mod r$; otherwise, Fail. The probability of returning the correct result is 1/2.

Before proceeding, we simplify $|\psi_2\rangle$ as most as possible using the periodicity of function f. Since the image of f on all points $(x - \ell s, \ell)$ for $0 \leq \ell < r$ is equal to the image of $(x, 0)$, we can collect the third register as

$$|\psi_2\rangle = \frac{1}{r} \sum_{x=0}^{r-1} \left(\sum_{\ell=0}^{r-1} |x - \ell s\rangle |\ell\rangle \right) |f(x, 0)\rangle.$$

In step 4, we measure the third register in the computational basis, obtaining an image of one of the representatives of the cosets of L in $\mathbb{Z}_r^2$ as follows

$$|\psi_3\rangle = \frac{1}{\sqrt{r}} \left(\sum_{\ell=0}^{r-1} |x_0 - \ell s\rangle |\ell\rangle \right) |f(x_0, 0)\rangle.$$

In step 5, we apply $F_r^\dagger \otimes F_r^\dagger$ to the first and second registers (we disregard the third register) obtaining

$$|\psi_4\rangle = \frac{1}{\sqrt{r}} \sum_{\ell=0}^{r-1} \left(F_r^\dagger |x_0 - \ell s\rangle \right) \left(F_r^\dagger |\ell\rangle \right).$$

We now simplify $|\psi_4\rangle$ as most as possible. Using the definition of the Fourier transform $F_r^\dagger$, we obtain

$$|\psi_4\rangle = \frac{1}{\sqrt{r}} \sum_{\ell=0}^{r-1} \left(\frac{1}{\sqrt{r}} \sum_{x=0}^{r-1} \omega_r^{x(x_0-\ell s)} |x\rangle \right) \left(\frac{1}{\sqrt{r}} \sum_{y=0}^{r-1} \omega_r^{y\ell} |y\rangle \right),$$

where $\omega_r = \exp(2\pi \mathrm{i}/r)$. Changing the order of the sums by pushing $\sum_{x,y}$ to the left, and by pushing $\sum_\ell$ and the term $\omega_r^{-x\ell s}$ to the second register, we obtain

$$|\psi_4\rangle = \frac{1}{r\sqrt{r}} \sum_{x,y=0}^{r-1} \omega_r^{xx_0} |x\rangle \sum_{\ell=0}^{r-1} \omega_r^{\ell(-xs+y)} |y\rangle.$$

Using that

$$\frac{1}{r} \sum_{\ell=0}^{r-1} \left(\omega_r^{-xs+y}\right)^\ell = \begin{cases} 1, & \text{if } y = xs, \\ 0, & \text{otherwise,} \end{cases}$$

we obtain

$$|\psi_4\rangle = \frac{1}{\sqrt{r}} \sum_{x,y=0}^{r-1} \omega_r^{xx_0} |x\rangle \, \delta_{y,xs} |y\rangle,$$

and by simplifying the sum over y, we obtain

$$|\psi_4\rangle = \frac{1}{\sqrt{r}} \sum_{x=0}^{r-1} \omega_r^{xx_0} |x\rangle \, |xs\rangle.$$

In step 6, we measure the first and second registers in the computational basis and obtain two results: r_1 and $r_2 = r_1 s$, where r_1 is chosen in the interval $[0, r-1]$ uniformly at random.

In step 7, if $\gcd(r_1, r) = 1$, we calculate $s \equiv r_2/r_1 \mod r$. Since $s \leq r < N$, s satisfies $a^s \equiv b \mod N$. For any odd r_1, $\gcd(r_1, r) = 1$. Since half of the values of r_1 is odd, the success probability is $1/2$.

Chapter 9

Grover's Algorithm

Grover's algorithm [23, 24] is a search algorithm initially developed for unstructured data. It can also be described in terms of an oracle, which is a function with some promise or property that can be evaluated as many times as we want, and our goal is to determine the property that the function has. This Chapter follows the latter description with a focus on the circuit model. Grover's algorithm is optimal, that is, it cannot be improved [5, 69], has already been used to create new quantum algorithms [33, 35], and is described in many books [3, 4, 6, 26, 28, 30, 32, 34, 40, 45, 49, 54, 63, 65].

9.1 Problem formulation in terms of an oracle

Let N be a power of 2 for some integer n, that is, $N = 2^n$. Suppose that $f : \{0, \ldots, N-1\} \to \{0, 1\}$ is a Boolean function such that $f(x) = 1$ if and only if $x = x_0$ for some fixed value x_0, that is,

$$f(x) = \begin{cases} 1, & \text{if } x = x_0, \\ 0, & \text{otherwise.} \end{cases}$$

Suppose that x_0 is unknown to us. How can we find x_0 by evaluating f? From a computational point of view, we want to evaluate f as few as possible.

Classically, the most efficient algorithm queries this function N times in the worst case, that is, the complexity of the classical algorithm in terms of the number of queries is $O(N)$. How is it done in practice? We ask someone else to generate a n-bit random number x_0. This person hides x_0 from us and makes a compiled subroutine of f. We can use the subroutine as many times as we want, but we cannot hack the code in search for x_0. The classical algorithm that solves this problem is an iteration that queries $f(x)$ for x from 0 to $2^n - 1$. As soon as $f(x)$ is 1, the program returns x_0.

Quantumly, it is possible to improve the query complexity to $O(\sqrt{N})$. How is it done in practice? We have to ask someone again to generate a n-bit random number x_0 and define f. This person, the *oracle*, implements f through a unitary matrix U_f in a quantum computer. We can use U_f, but we cannot see the details of the implementation of U_f. Each time we use U_f, we add a unit to the count. We can use additional gates that obviously don't depend on x_0.

We have the same problem that can be solved by algorithms executed in two different machines. In the first case, a classical computer with $O(n)$ bits is used and the solution is found

after $O(N)$ steps. In the second case, a quantum computer with $O(n)$ qubits is used and the solution is found after $O(\sqrt{N})$ steps. This gain in complexity justifies the investment in quantum hardware and the study of techniques to develop quantum algorithms, which necessarily have to use the state superposition. In the case of Grover's algorithm, U_f acting on a superposition of states evaluates f at more than one point in the domain at the same time with the same available resources. The classical computer needs an exponential number of processors to perform this task with the same efficiency.

9.2 How to implement the oracle on a quantum computer

The first step in developing a quantum algorithm that solves Grover's problem is the implementation of the function f. Since f is a Boolean function whose truth table has a single row with output 1, f can be implemented with a multiqubit Toffoli gate activated by x_0, as described in Sec. 2.7 on Page 23. This gate has an associated unitary matrix U_f, which is defined by its action on the computational basis as

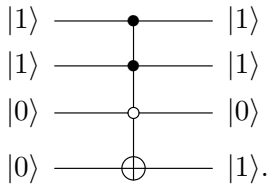
$$U_f|x\rangle|i\rangle = |x\rangle|i \oplus f(x)\rangle,$$

where x is a n-bit string and i is a bit. The first register has n qubits and the second register has one qubit. Note that if we take $i = 0$, the above equation reduces to

$$U_f|x\rangle|0\rangle = \begin{cases} |x_0\rangle|1\rangle, & \text{if } x = x_0, \\ |x\rangle|0\rangle, & \text{otherwise,} \end{cases}$$

which describes the output of a multiqubit Toffoli gate activated by x_0 (and only by x_0) when the input is $|x\rangle|0\rangle$. The result of the calculation of $f(x)$ is stored in the second register while the state of the first register remains unchanged.

For example, the circuit that implements U_f in the case $N = 8$ and $x_0 = 6$, that is, $f(110) = 1$ and $f(j) = 0$ if $j \neq 110$, is



The first register has three qubits. Note that the state of the second register changes from $|0\rangle$ to $|1\rangle$ only if the input to the first register is $|110\rangle$ because 110 activates the three controls and all other 3-bit strings do not.

In Grover's algorithm, the state of the second register is always

$$|-\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}}.$$

Using linearity, the action of U_f is given by

$$U_f|x\rangle|-\rangle = \begin{cases} -|x_0\rangle|-\rangle, & \text{if } x = x_0, \\ |x\rangle|-\rangle, & \text{otherwise.} \end{cases}$$

The same result is obtained from Proposition 3.1 on Page 28, which states that

$$U_f|x\rangle|-\rangle = (-1)^{f(x)}|x\rangle|-\rangle.$$

9.3 The algorithm

Grover's algorithm uses an additional operator defined as

$$G = \left(2\,|\mathrm{d}\rangle\langle\mathrm{d}| - I_N\right) \otimes I_2,$$

where

$$|\mathrm{d}\rangle = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} |j\rangle.$$

The notation "$|\mathrm{d}\rangle\langle\mathrm{d}|$" is called *external product* between the vector $|\mathrm{d}\rangle$ (a $N \times 1$ matrix) and the dual vector $\langle\mathrm{d}|$ (a $1 \times N$ matrix). The external product is equivalent to the matrix product. Multiplying a $N \times 1$ matrix by a $1 \times N$ matrix results in a $N \times N$ matrix. Therefore, $|\mathrm{d}\rangle\langle\mathrm{d}|$ is a $N \times N$ matrix, given by

$$|\mathrm{d}\rangle\langle\mathrm{d}| = \frac{1}{N} \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & 1 & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \cdots & 1 \end{bmatrix},$$

and

$$\left(2\,|\mathrm{d}\rangle\langle\mathrm{d}| - I_N\right) = \frac{1}{N} \begin{bmatrix} (2-N) & 2 & \cdots & 2 \\ 2 & (2-N) & \cdots & 2 \\ \vdots & \vdots & \ddots & \vdots \\ 2 & 2 & \cdots & (2-N) \end{bmatrix}$$

which is called Grover matrix (or Grover operator).

Grover's algorithm is described in Algorithm 9.1.

Algorithm 9.1: Grover's algorithm

Input: An integer N and a function $f : \{0, ..., N-1\} \to \{0, 1\}$ such that $f(x) = 1$ only for one point $x = x_0$ in the domain.

Output: x_0 with probability equal to or greater than $1 - \frac{1}{N}$.

1 Prepare the initial state $|\mathrm{d}\rangle|-\rangle$ using $n+1$ qubits;

2 Apply $(G\,U_f)^t$, where $t = \left\lfloor \frac{\pi}{4}\sqrt{N} \right\rfloor$;

3 Measure the first register in the computational basis.

9.4 Non-economical circuit of Grover's algorithm

The goal of this Section is to find the circuit that implements the Grover operator using our knowledge of implementing Boolean functions. The circuit uses more qubits than necessary, but later we show how to obtain an economical version of the circuit.

To obtain the circuit, we have to do an algebraic manipulation with the expression of the Grover matrix $\left(2\,|\mathrm{d}\rangle\langle\mathrm{d}| - I_N\right)$. Note that

$$|\mathrm{d}\rangle = H^{\otimes n}|0\rangle$$

where $|0\rangle$ is in the decimal notation and $H^{\otimes n} = H \otimes \cdots \otimes H$. Transposing the above equation, we obtain

$$\langle \mathrm{d}| = \langle 0|H^{\otimes n}.$$

Using $(H^{\otimes n}) \cdot (H^{\otimes n}) = (H \cdot H)^{\otimes n} = (I_2)^{\otimes n} = I_N$, we obtain

$$\big(2\,|\mathrm{d}\rangle\langle \mathrm{d}| - I_N\big) = H^{\otimes n}\big(2\,|0\rangle\langle 0| - I_N\big)H^{\otimes n},$$

where

$$\big(2\,|0\rangle\langle 0| - I_N\big) = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & -1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & -1 \end{bmatrix}.$$

Matrix $\big(2\,|0\rangle\langle 0| - I_N\big)$ acts only on the first register. However, it is simpler to implement this matrix using both registers. Let us show that it is implemented by a multiqubit Toffoli gate activated by 0. Indeed, the action of $\big(2\,|0\rangle\langle 0| - I_N\big)$ on $|x\rangle$, where x is a n-bit string, is

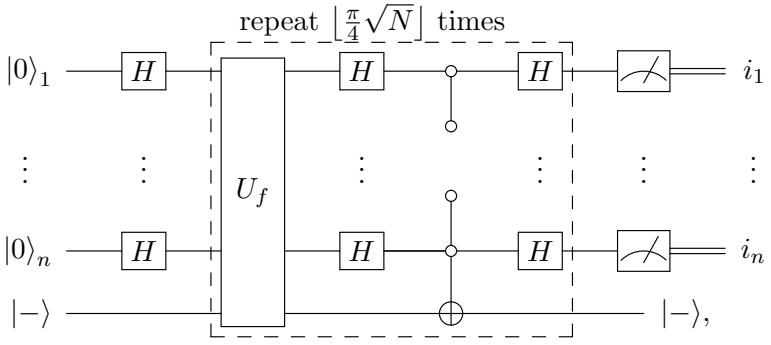
$$\big(2\,|0\rangle\langle 0| - I_N\big)|x\rangle = \begin{cases} |0\rangle, & \text{if } x = 0, \\ -|x\rangle, & \text{otherwise.} \end{cases}$$

Therefore, the action of $\big(2\,|0\rangle\langle 0| - I_N\big)$ on the first register is the same as the action of $(-U_{f'})$ on both registers when $x_0 = 0$ (the state of the second register must be $|-\rangle$), where

$$f'(x) = \begin{cases} 1, & \text{if } x = 0, \\ 0, & \text{otherwise.} \end{cases}$$

The minus sign in $(-U_{f'})$ changes neither the result of the algorithm nor the final probability. That is, using G or $-G$ in Grover's algorithm does not change the final result.

Using these algebraic results, we conclude that a circuit that implements Grover's algorithm is

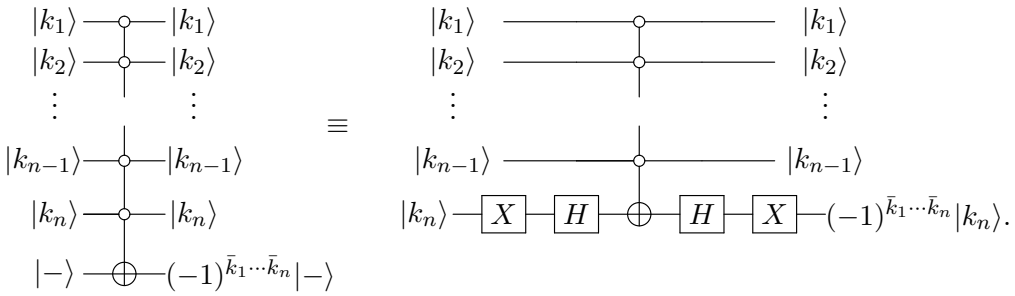


where bits i_1, ..., i_n are the outputs of the measurements. Those bits are the bit of x_0, that is $x_0 = (i_1 \ldots i_n)_2$, with high probability.

9.5 Economical circuit of Grover's algorithm

The second register of Grover's algorithm can be discarded since it is possible to make a more economical implementation of the oracle [36]. Let us start by showing how to implement the

operator $\left(2\left|0\right\rangle\left\langle 0\right| - I_N\right)$ (modulo a global phase), which enables us to implement operator G using only the first register. Let us show the equivalence of the following circuits:



The output of the left-hand circuit is the $(n+1)$-qubit state

$$|k_1\rangle \otimes \cdots \otimes |k_n\rangle \otimes \left((-1)^{\bar{k}_1\cdots\bar{k}_n}|-\rangle\right),$$

which is obtained from the definition of the multiqubit Toffoli gate activated only when qubits $k_1, \ldots, k_n$ are set to 0. The Kronecker product has the property

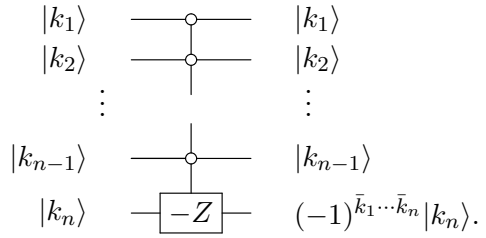
$$|v_1\rangle \otimes (a|v_2\rangle) = (a|v_1\rangle) \otimes |v_2\rangle = a(|v_1\rangle \otimes |v_2\rangle),$$

for any vectors $|v_1\rangle$, $|v_2\rangle$ and any scalar a. Then, we move the scalar term $(-1)^{\bar{k}_1\cdots\bar{k}_n}$ to the first register whose output is

$$|k_1\rangle \otimes \cdots \otimes |k_{n-1}\rangle \otimes \left((-1)^{\bar{k}_1\cdots\bar{k}_n}|k_n\rangle\right) \;=\; (-1)^{\bar{k}_1\cdots\bar{k}_n}|k_1\rangle \otimes \cdots \otimes |k_n\rangle.$$

The state of the second register is $|-\rangle$ and this register will be discarded at the end of the process.

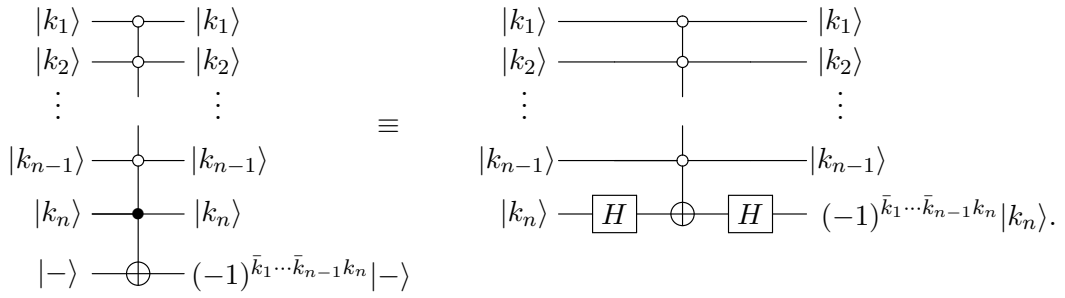
Now let us show that the output of the right-hand circuit is the same. We use the fact that $XHXHX = -Z$, therefore the right-hand circuit is equivalent to



The output is obtained using that $(-Z)|k_n\rangle = (-1)^{\bar{k}_n}|k_n\rangle$, and since $-Z$ is activated only when qubits $k_1, \ldots, k_{n-1}$ are set to 0, we obtain the overall output $(-1)^{\bar{k}_1\cdots\bar{k}_{n-1}\bar{k}_n}|k_1\cdots k_{n-1}k_n\rangle$. In conclusion, the result of the first circuit (after the elimination of the second register) is the same as the result of the second circuit. Since we are going to perform a measurement of the first register only, this completes the proof that we can replace *without any loss* the first circuit with the second in Grover's algorithm. This replacement is not valid in all algorithms. So far we have shown that G (modulo a global phase) can be implemented using only the first register.

Let us consider the oracle. If the oracle chooses $x_0 = 0$, the circuit of U_f is a multiqubit Toffoli gate activated only when all qubits of the first register are set to 0. In this case, we

have already shown how to implement U_f using only the first register. The oracle would use the right-hand circuit depicted in the beginning of this Section. If the oracle choose $x_0 = 1$, the circuit of U_f is a multiqubit Toffoli gate that is activated only when all qubits of the first register are set to 0 except the n-th qubit, which is set to 1. In this case, we have the following circuit equivalence:



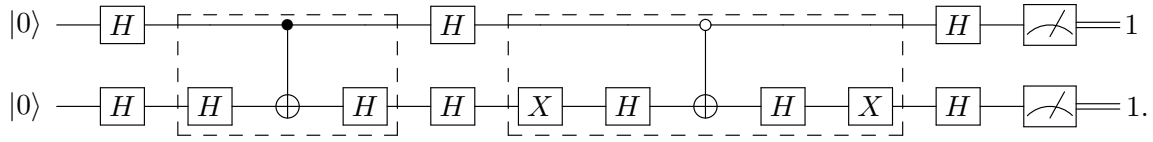
The equivalence check is similar to the previous case but now it is obtained using that $HXH = Z$ and $Z|k_n\rangle = (-1)^{k_n}|k_n\rangle$, and since Z is activated only when qubits $k_1, k_2, \ldots, k_{n-1}$ are set to 0, we obtain the overall output $(-1)^{\bar{k}_1\bar{k}_2\cdots\bar{k}_{n-1}k_n}|k_1k_2\cdots k_n\rangle$. This concludes the proof that the oracle would use a n-qubit circuit if $x_0 = 1$.

The remaining cases, $x_0 \geq 2$, are obtained from the previous results. If the rightmost (last) bit of x_0 is 0, we use the circuit equivalence described at the beginning of this Section in the following way: If any control (except the n-th) on the left-hand circuit changes from empty to full circle, the same must happen to the controls on the right-hand circuit. If the rightmost bit of x_0 is 1, we use the second circuit equivalence of this Section, and in the same fashion if any control (except the n-th) on the left-hand circuit changes from empty to full circle, the same must happen to the controls on the right-hand circuit. The expression of the output changes accordingly.

Thus, not only G but also U_f can be implemented with n qubits by eliminating the second register and by introducing two Hadamard gates plus two Pauli X gates if the n-th qubit is activated by 0, and by introducing only two Hadamard gates if the n-th qubit is activated by 1.

Economical circuit when $N = 4$

The circuit of Grover's algorithm in the economical form when $N = 4$ and $x_0 = 11$ is

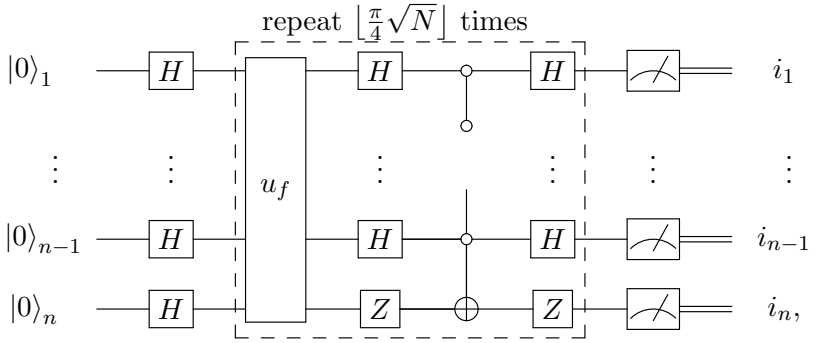


The gates inside the first dashed box implement the oracle and the gates inside the second dashed box implement $\left(2\,|0\rangle\langle 0| - I_N\right)$ (modulo a global phase). This circuit can be simplified by substituting HXH in the second qubit with Z in two places.

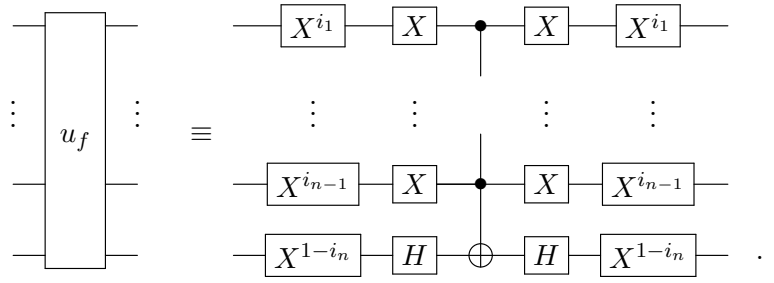
The goal of Grover's algorithm is to determine x_0 by querying the oracle, that is, using the first dashed box, without looking at its implementation details. We have to pretend that the first dashed box is a black box. When $N = 4$, there are four possible black boxes, case $x_0 = 11$ is one of them. The gates that implement the oracle when $x_0 = 00$, $x_0 = 01$, and $x_0 = 10$ are $(X\otimes XH)\mathrm{CNOT}(X\otimes HX)$, $(X\otimes H)\mathrm{CNOT}(X\otimes H)$, and $(I\otimes XH)\mathrm{CNOT}(I\otimes HX)$, respectively. When $N = 4$, the output is the correct one with probability 1.

Economical circuit for arbitrary N

For an arbitrary N ($N = 2^n$ and $n \geq 2$), the circuit of Grover's algorithm with only n qubits is



where the matrix u_f is the economical version of U_f and the circuit for u_f for an arbitrary $x_0 = (i_1 \dots i_n)_2$ is



9.6 Analysis of Grover's algorithm

Why does Grover's algorithm work correctly? We answer this question using the economical form of the algorithm. The operators in this case are u_f, which is defined as

$$u_f = \sum_x (-1)^{f(x)} |x\rangle\langle x|,$$

and

$$g = 2\,|\mathrm{d}\rangle\langle\mathrm{d}| - I_N.$$

Operator u_f is the economical version of U_f, that is, u_f is a N-dimensional operator whose action on the computational basis is

$$u_f|x\rangle = \begin{cases} -|x_0\rangle, & \text{if } x = x_0, \\ |x\rangle, & \text{otherwise.} \end{cases}$$

In its turn, g is the economical version of operator G. The economical version of Grover's algorithm is described in Algorithm 9.2.

The goal of the algorithm is to find x_0, which is a n-bit string. It will be accomplished if the state of the qubits just before the measurement is $|x_0\rangle$ because the measurement in this case returns x_0. The analysis of the algorithm that we now start to describe is based on a geometric

Algorithm 9.2: Grover's algorithm (economical version)

Input: An integer N (power of 2) and a function $f : \{0, ..., N-1\} \to \{0, 1\}$ such that $f(x) = 1$ only for one point $x = x_0$ in the domain.

Output: x_0 with probability equal to or greater than $1 - \frac{1}{N}$.

1 Prepare the initial state $|\mathrm{d}\rangle$ using n qubits ;

2 Apply $(g\, u_f)^t$, where $t = \left\lfloor \frac{\pi}{4}\sqrt{N} \right\rfloor$;

3 Measure all qubits in the computational basis.

interpretation of vector reflections [1]. At the beginning of the algorithm, the state of the qubits is $|\mathrm{d}\rangle$. For large N, $|\mathrm{d}\rangle$ is almost orthogonal to $|x_0\rangle$. Fig. 9.1 shows vectors $|\mathrm{d}\rangle$ and $|x_0\rangle$, where $\theta/2$ is the angle between $|\mathrm{d}\rangle$ and the horizontal axis. Any other representation of those vectors can be used in the analysis of the algorithm provided that $|\mathrm{d}\rangle$ is almost orthogonal to $|x_0\rangle$.

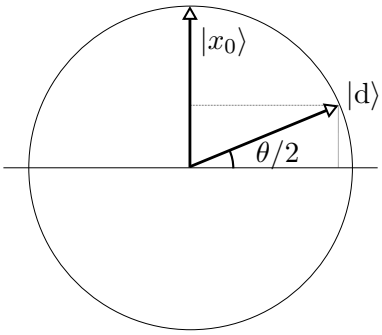


Figure 9.1: Depiction of vectors $|x_0\rangle$ and $|\mathrm{d}\rangle$.

The angle θ is very small for large N, and in this case, $\theta/2$ is a good approximation of $\sin(\theta/2)$. In addition, the sine of an angle is equal to the cosine of the complement, that is,

$$\frac{\theta}{2} \approx \sin\frac{\theta}{2} = \cos\left(\frac{\pi}{2} - \frac{\theta}{2}\right).$$

As $(\pi - \theta)/2$ is the angle between $|x_0\rangle$ and $|\mathrm{d}\rangle$, by definition of the inner product, $\cos(\pi - \theta)/2$ is the inner product of vectors $|x_0\rangle$ and $|\mathrm{d}\rangle$, the result of which is

$$\frac{\theta}{2} \approx \sin\frac{\theta}{2} = \cos\left(\frac{\pi}{2} - \frac{\theta}{2}\right) = \langle x_0|\mathrm{d}\rangle = \frac{1}{\sqrt{N}}.$$

Therefore,

$$\theta \approx \frac{2}{\sqrt{N}}.$$

The first step of Algorithm 9.2 is the preparation of the initial state $|\mathrm{d}\rangle$. The next step is to apply u_f to $|\mathrm{d}\rangle$. The action of u_f on $|\mathrm{d}\rangle$ (written in the computational basis) inverts the sign of the amplitude of $|x_0\rangle$ and does not change the other amplitudes. The amplitude of $|x_0\rangle$ is the orthogonal projection of $|\mathrm{d}\rangle$ on the vertical axis—see Fig. 9.1, which is inverted by the action of u_f. Geometrically, the action of u_f is represented by a reflection of $|\mathrm{d}\rangle$ about the horizontal axis. The angle between the vectors $|\mathrm{d}\rangle$ and $(u_f|\mathrm{d}\rangle)$ is θ, as shown in Fig. 9.2.

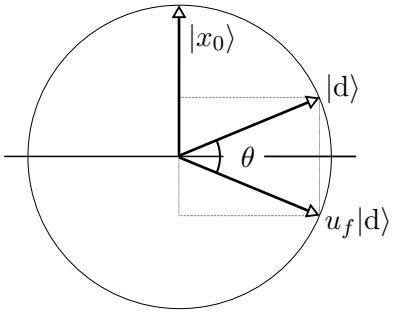


Figure 9.2: Vector $u_f|\mathrm{d}\rangle$ is a reflexion of $|\mathrm{d}\rangle$ about the horizontal axis.

The next step is to apply $g = \big(2\,|\mathrm{d}\rangle\langle\mathrm{d}| - I_N\big)$. Let us show that the action of g is a reflection about the axis defined by $|\mathrm{d}\rangle$. This proof is done in two parts. First, we show that $|\mathrm{d}\rangle$ is invariant under the action of g. Second, we show that the action of g on $\big|\mathrm{d}^\perp\big\rangle$ inverts the sign of $\big|\mathrm{d}^\perp\big\rangle$, where $\big|\mathrm{d}^\perp\big\rangle$ is any vector orthogonal to $|\mathrm{d}\rangle$. The first step follows from

$$g|\mathrm{d}\rangle = \big(2\,|\mathrm{d}\rangle\langle\mathrm{d}| - I_N\big)|\mathrm{d}\rangle = 2\,|\mathrm{d}\rangle\,\big\langle\mathrm{d}\big|\mathrm{d}\big\rangle - |\mathrm{d}\rangle = |\mathrm{d}\rangle,$$

because $\big\langle\mathrm{d}\big|\mathrm{d}\big\rangle = 1$. The second step follows from

$$g\Big|\mathrm{d}^\perp\Big\rangle = \big(2\,|\mathrm{d}\rangle\langle\mathrm{d}| - I_N\big)\Big|\mathrm{d}^\perp\Big\rangle = 2\,|\mathrm{d}\rangle\,\Big\langle\mathrm{d}\Big|\mathrm{d}^\perp\Big\rangle - \Big|\mathrm{d}^\perp\Big\rangle = -\Big|\mathrm{d}^\perp\Big\rangle,$$

because $\big\langle\mathrm{d}\big|\mathrm{d}^\perp\big\rangle = 0$.

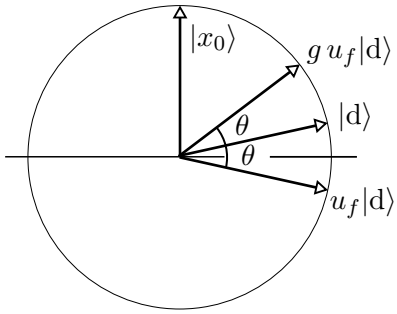


Figure 9.3: Vector $g\,u_f|\mathrm{d}\rangle$ is a reflexion of $u_f|\mathrm{d}\rangle$ about $|\mathrm{d}\rangle$.

Fig. 9.3 depicts $g\,u_f|\mathrm{d}\rangle$ and shows that the action of gu_f rotates the initial state by θ degrees towards $|x_0\rangle$. Since θ is a small angle, this achievement is modest but promising. It is easy to see that the second action of gu_f repeats the process of rotating θ degrees towards $|x_0\rangle$. We want to know how many iterations r are needed such as $r\theta = \pi/2$? The number of iterations is

$$r = \left\lfloor \frac{\pi}{2\theta} \right\rfloor = \left\lfloor \frac{\pi}{4}\sqrt{N} \right\rfloor.$$

It is still missing the calculation of the success probability. After r iterations, the state of the qubits is

$$|\psi\rangle = (g\,u_f)^{\left\lfloor \frac{\pi}{4}\sqrt{N} \right\rfloor}|\mathrm{d}\rangle.$$

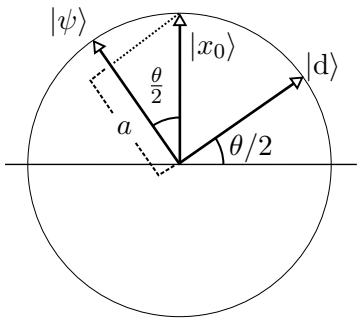


Figure 9.4: Vector $|\psi\rangle$ is the final state before measurement and a is the norm of the projection of $|x_0\rangle$ on $|\psi\rangle$. The angle between $|\psi\rangle$ and $|x_0\rangle$ is less than or equal to $\theta/2$.

Vector $|\psi\rangle$ is almost orthogonal to $|\mathrm{d}\rangle$ at this point, as depicted in Fig. 9.4. The angle between $|\psi\rangle$ and $|x_0\rangle$ is less than or equal to $\theta/2$. The success probability is greater than or equal to the absolute square of the amplitude of $|x_0\rangle$ in the decomposition of $|\psi\rangle$ in the computational basis. This amplitude is a as shown in Fig. 9.4. The orthogonal projection of $|x_0\rangle$ on the final state is at most $\cos(\theta/2)$. Therefore, the success probability $p = |a|^2$ satisfies

$$p \;\geq\; \cos^2\frac{\theta}{2} \;\geq\; 1 - \sin^2\frac{\theta}{2} \;\geq\; 1 - \frac{1}{N}.$$

The case $N = 4$ case is special because $\theta = 60°$, since $\sin(\theta/2) = 1/\sqrt{N}$. With one application of gu_f, vector $|\mathrm{d}\rangle$ rotates 60° and coincides with $|x_0\rangle$. In this case, the success probability is exactly $p = 1$.

9.7 Final remarks

The same technique for implementing the oracle using n qubits analyzed in this Chapter can be applied to the implementation of the Deutsch-Jozsa algorithm with only n qubits. Note that the state of the second register in the Deutsch-Jozsa circuit before applying U_f is $|-\rangle$. What we have to do is to discard the second register and to replace U_f in the Deutsch-Jozsa circuit by u_f described at the end of Sec. 9.5.

Chapter 10

Phase Estimation and Applications

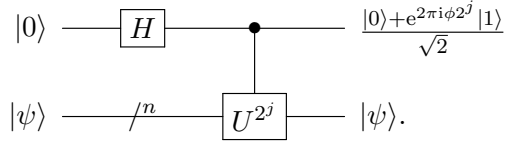
Kitaev published the quantum phase estimation algorithm as a preprint in 1995 [31], and later as a section in a book in Russian, which was translated into English [32]. Kitaev's method is based on a procedure for measuring an eigenvalue of a unitary operator, that is, given a unitary operator U and one of its eigenvectors $|\psi\rangle$, the algorithm finds the eigenvalue $\exp(2\pi\phi)$, so that $U|\psi\rangle = \exp(2\pi\phi)|\psi\rangle$, where ϕ is the phase of the eigenvalue. This algorithm provides an alternative way of factoring integers and calculating discrete logarithms. Not only that, it is used in many applications such as quantum counting. This algorithm has been described in many books [4, 6, 27, 30, 45, 63].

10.1 Quantum phase estimation algorithm

Suppose we have a n-qubit unitary operator U and we know one of its eigenvectors $|\psi\rangle$. We do not know the eigenvalue associated with $|\psi\rangle$, but we know that its analytical expression is $\mathrm{e}^{2\pi\mathrm{i}\phi}$, where $0 \leq \phi < 1$ (ϕ is unknown), because U is unitary. We assume for now that $\phi = 0.\phi_1 \cdots \phi_m$ for some integer m, where ϕ_1, ..., ϕ_m are bits, that is, the phase of the eigenvalue $\mathrm{e}^{2\pi\mathrm{i}\phi}$ is a rational multiple of 2π. The goal of the phase estimation algorithm is to determine ϕ using U as an oracle and $|\psi\rangle$ as an input.

Basic block

The basic block of the circuit of the quantum phase estimation algorithm depends on an integer $0 \leq j < m$ and is given by



To verify the correctness of the output of the basic block, we have to use

$$U|\psi\rangle = \mathrm{e}^{2\pi\mathrm{i}\phi}|\psi\rangle,$$

and also

$$U^{2^j}|\psi\rangle = \mathrm{e}^{2\pi\mathrm{i}\phi\,2^j}|\psi\rangle.$$

The output of the basic block is obtained by applying the controlled U^{2^j} operator on $(H|0\rangle)\otimes|\psi\rangle$, that is,

$$C\big(U^{2^j}\big)\left(\frac{|0\rangle|\psi\rangle+|1\rangle|\psi\rangle}{\sqrt{2}}\right) \;=\; \frac{|0\rangle|\psi\rangle+|1\rangle U^{2^j}|\psi\rangle}{\sqrt{2}} \;=\; \frac{|0\rangle+\mathrm{e}^{2\pi\mathrm{i}\phi 2^j}|1\rangle}{\sqrt{2}}\otimes|\psi\rangle.$$

Here we see an example of the *phase kickback process* because the phase was produced by the action of U on the second register but it appears as a relative phase of the first qubit after $|\psi\rangle$ has been collected.

There is an alternative way of writing the eigenvalue of U^{2^j} associated with $|\psi\rangle$. Using that $\phi = 0.\phi_1\cdots\phi_m$ in binary, then

$$\phi \;=\; \frac{\phi_1}{2}+\frac{\phi_2}{2^2}+\cdots+\frac{\phi_m}{2^m}.$$

Multiplying by 2^j, we obtain

$$\phi\, 2^j \;=\; 2^{j-1}\phi_1+\cdots+2\phi_{j-1}+\phi_j+\frac{\phi_{j+1}}{2}+\cdots+\frac{\phi_m}{2^{m-j}}.$$

It is straightforward to check that

$$\exp\big(2\pi\mathrm{i}\phi\, 2^j\big) \;=\; \exp\left(2\pi\mathrm{i}\left(\frac{\phi_{j+1}}{2}+\cdots+\frac{\phi_m}{2^{m-j}}\right)\right)$$

because $\exp\big(2\pi\mathrm{i}\, 2^{j-1}\phi_1\big)=\cdots=\exp\big(2\pi\mathrm{i}\phi_j\big)=1$. Then,

$$\exp\big(2\pi\mathrm{i}\phi\, 2^j\big) \;=\; \exp\left(2\pi\mathrm{i}\, 0.\phi_{j+1}\cdots\phi_m\right).$$

Note that the first digits of ϕ were eliminated.

The implementation of U^{2^j} is not necessarily performed by 2^j applications of U. This method is inefficient if m is large. The implementation depends on specific applications of the phase estimation algorithm. For instance, if U performs modular arithmetic, the *repeated squaring method* is employed.

The circuit of the quantum phase estimation algorithm has two blocks. The first is made of m basic blocks and the second is the inverse Fourier transform. Let us start by describing the first block.

First block

The circuit of first block comprises m basic blocks with a common second register as depicted in Fig. 10.1. The first register has m qubits with input $|0\rangle^{\otimes m}$ and the second register has n qubits with input $|\psi\rangle$. The output is a direct consequence of each basic block, which uses U^{2^j}, where j runs from 0 to $m-1$. The order of the controlled operations is irrelevant but j must be 0 for the m-th qubit, j must be 1 for the $(m-1)$-th qubit, and so on.

The output of the first register of the first block is

$$\frac{|0\rangle+\mathrm{e}^{2\pi\mathrm{i}\phi 2^{m-1}}|1\rangle}{\sqrt{2}}\otimes\frac{|0\rangle+\mathrm{e}^{2\pi\mathrm{i}\phi 2^{m-2}}|1\rangle}{\sqrt{2}}\otimes\cdots\otimes\frac{|0\rangle+\mathrm{e}^{2\pi\mathrm{i}\phi 2^{0}}|1\rangle}{\sqrt{2}}.$$

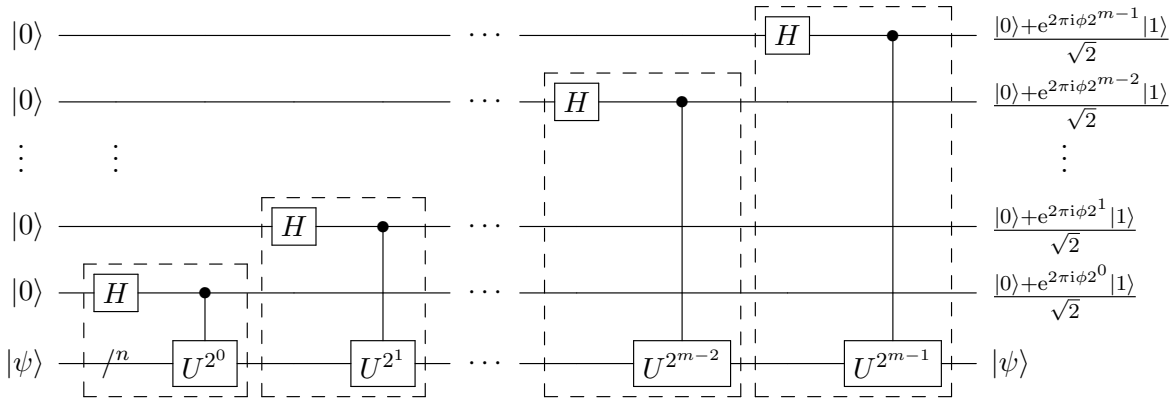


Figure 10.1: The first block of the phase estimation circuit is made of m basic blocks sharing the second register.

This output can be simplified into a very neat expression. In order to do so, let us replace each term with an equivalent term using a binary sum and collecting the denominators

$$\frac{1}{\sqrt{2^m}} \sum_{\ell_1=0}^{1} e^{2\pi i \phi 2^{m-1} \ell_1} |\ell_1\rangle \otimes \sum_{\ell_2=0}^{1} e^{2\pi i \phi 2^{m-2} \ell_2} |\ell_2\rangle \otimes \cdots \otimes \sum_{\ell_1=0}^{m} e^{2\pi i \phi 2^{0} \ell_m} |\ell_m\rangle.$$

Pushing all sums to the beginning of the expression and combining all exponentials, we obtain

$$\frac{1}{\sqrt{2^m}} \sum_{\ell_1,\ldots,\ell_m=0}^{1} e^{2\pi i \phi (2^{m-1}\ell_1 + \cdots + 2^0 \ell_m)} |\ell_1\rangle \otimes \ldots \otimes |\ell_m\rangle.$$

Converting the binary into decimal, we obtain

$$\frac{1}{\sqrt{2^m}} \sum_{\ell=0}^{2^m-1} e^{2\pi i \, \phi \, \ell} |\ell\rangle.$$

This is the neat expression we were looking for. Let us summarize the first block:

$$|0\rangle^{\otimes m} \otimes |\psi\rangle \xrightarrow[\text{block}]{\text{first}} \left(\frac{1}{\sqrt{2^m}} \sum_{\ell=0}^{2^m-1} e^{2\pi i \, \phi \, \ell} |\ell\rangle \right) \otimes |\psi\rangle,$$

where m is the number of qubits of the first register, $|\psi\rangle$ is an eigenvector of U with eigenvalue $\exp(2\pi i \, \phi)$, and $\phi = 0.\phi_1 ... \phi_m$. In the next Subsection, we show that the output of the first register is

$$F_{2^m} |\phi_1, \ldots, \phi_m\rangle,$$

where F_{2^m} is the Fourier transform, defined in Sec. 7.4.

Full circuit of the quantum phase estimation (QPE)

Now we show that the second block of the QPE algorithm is the inverse Fourier transform. In the last Subsection, we have shown that the output of the first register of the first block is

$$\frac{1}{\sqrt{2^m}} \sum_{\ell=0}^{2^m-1} e^{2\pi i \, \phi \, \ell} |\ell\rangle.$$

On the other hand, the action of the Fourier transform F_{2^m} on a generic state $|j\rangle$ of the computation basis is

$$F_{2^m}|j\rangle \;=\; \frac{1}{\sqrt{2^m}} \sum_{\ell=0}^{2^m-1} \mathrm{e}^{\frac{2\pi \mathrm{i} j \ell}{2^m}} |\ell\rangle.$$

Inverting the equation, taking $j = (\phi_1 ... \phi_m)_2 = (2^m \phi)_{10}$ and $|j\rangle = |\phi_1\rangle \otimes ... \otimes |\phi_m\rangle$, we obtain

$$F^{\dagger}_{2^m} \left(\frac{1}{\sqrt{2^m}} \sum_{\ell=0}^{2^m-1} \mathrm{e}^{2\pi \mathrm{i}\, \phi\, \ell} |\ell\rangle \right) \;=\; |2^m \phi\rangle \;=\; |\phi_1\rangle \otimes \cdots \otimes |\phi_m\rangle.$$

If we apply the inverse Fourier transform to the output of the first block, the result is a state of the computational basis equal to $|\phi_1\rangle \otimes ... \otimes |\phi_m\rangle$. This means that a measurement in the computational basis reveals with certainty each fractional bit of ϕ because we are assuming that ϕ has been given with m bits. In the general case, the result of the algorithm is a good m-bit estimation of ϕ, which is denoted by $\tilde{\phi}$, that is, $\tilde{\phi} \approx \phi 2^m$.

The full circuit of the phase estimation algorithm is depicted in Fig. 10.2. The algorithm is described in Algorithm 10.1.

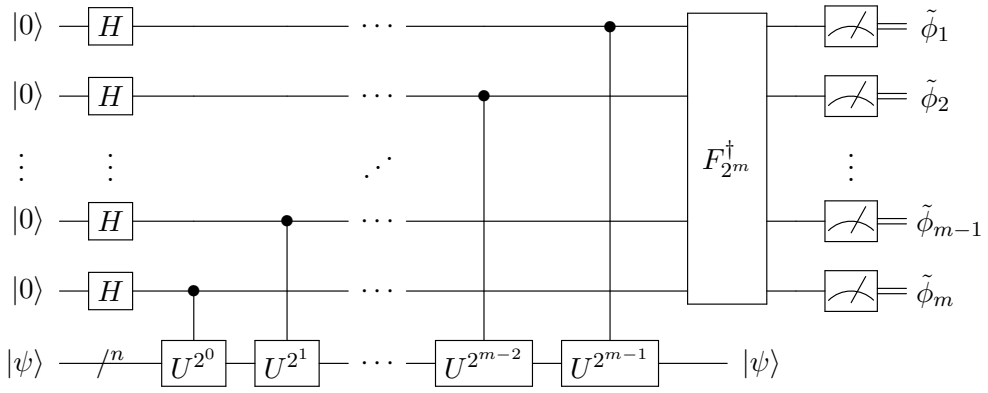


Figure 10.2: Full circuit of the QPE algorithm.

Algorithm 10.1: Quantum phase estimation algorithm

Input: Eigenvector $|\psi\rangle$ of U.

Output: Number $\tilde{\phi}$, where $\exp(2\pi \mathrm{i}\phi)$ is the eigenvalue of $|\psi\rangle$ and $\tilde{\phi} \approx \phi 2^m$.

1 Prepare the initial state $|0\rangle^{\otimes m} \otimes |\psi\rangle$;

2 Apply $H^{\otimes m}$ to the first register;

3 For ℓ in $[0, m-1]$ apply the controlled operation $C^{m-\ell}\left(U^{2^\ell}\right)$, where the control qubit is $m-\ell$ and the target is the 2nd register;

4 Apply $F^{\dagger}_{2^m}$ to the first register;

5 Measure the first register in the computational basis.

10.2 Application to order-finding

Let N and a be positive integers so that $1 < a < N$ and $\gcd(a, N) = 1$. The multiplicative order of a modulo N is the smallest positive integer r that obeys

$$a^r \equiv 1 \mod N.$$

Given a and N, the order-finding is the problem of calculating r. In this Section, we show how to solve the order-finding problem efficiently using the phase estimation algorithm providing an alternative to Shor's factoring algorithm.

The strategy is to replace $|\psi\rangle$ in Algorithm 10.1 by $|1\rangle$ (the second vector of the computational basis of the second register) and choose U as the unitary operator that multiplies the input by a modulo N, that is,

$$U|y\rangle = |ay \mod N\rangle. \tag{10.1}$$

The input is a vector $|y\rangle$ of the computational basis of the second register. We may think that y is represented in the decimal system. The output is also a vector $|y'\rangle$ of the computational basis of the second register, which is obtained by calculating $ay \equiv y'$ modulo N. U is a unitary operator because $\gcd(a, N) = 1$. $U^\dagger$ is defined accordingly using a^{-1} modulo N, that is,

$$U^\dagger|y\rangle = \left|a^{-1}y \mod N\right\rangle.$$

The motivation for using U here is that the repeated application of U produces successive powers of a, in fact, $U^j|y\rangle = \left|a^j y\right\rangle$. The number of qubits n of the second register must be able to lodge U, then we take $n = \lceil \log_2 N \rceil$.

In order to understand the order-finding as a phase estimation algorithm, let us find the eigenvectors of U. It is straightforward to obtain a 1-eigenvector because set $\{a^0, a^1, ...a^{r-1}\}$, where r is the order of a modulo N, is invariant under the multiplication by a. Then, the normalized vector

$$|\psi_0\rangle = \frac{1}{\sqrt{r}} \sum_{\ell=0}^{r-1} \left|a^\ell\right\rangle$$

is a 1-eigenvector of U. Since this is the first vector of the Fourier basis (F_r) when we consider the transformation of the subset $\left\{\left|a^0\right\rangle, \left|a^1\right\rangle, ...\left|a^{r-1}\right\rangle\right\}$ of the computational basis, let us define the remaining ones:

$$|\psi_k\rangle = \frac{1}{\sqrt{r}} \sum_{\ell=0}^{r-1} \mathrm{e}^{-\frac{2\pi \mathrm{i} k\ell}{r}} \left|a^\ell\right\rangle. \tag{10.2}$$

Now let us check that each $|\psi_k\rangle$ is an eigenvector of U. In fact,

$$\begin{aligned} U|\psi_k\rangle &= \frac{1}{\sqrt{r}} \sum_{\ell=0}^{r-1} \mathrm{e}^{-\frac{2\pi \mathrm{i} k\ell}{r}} \left|a^{\ell+1}\right\rangle \\ &= \frac{1}{\sqrt{r}} \sum_{\ell=0}^{r-1} \mathrm{e}^{-\frac{2\pi \mathrm{i} k(\ell-1)}{r}} \left|a^{\ell}\right\rangle \\ &= \mathrm{e}^{\frac{2\pi \mathrm{i} k}{r}} |\psi_k\rangle. \end{aligned}$$

We conclude that $|\psi_k\rangle$ is an eigenvector of U with eigenvalue $\exp(2\pi \mathrm{i} k/r)$ for $0 \le k < r$, whose phase is k/r If we are able to prepare the input to the second register of the phase estimation

algorithm as $|\psi_k\rangle$ for some k, we will obtain an approximation of k/r as the output, and then we find a candidate for the order of a using the continued fraction expansion. If the input to the second register is $|\psi_k\rangle$, the output of the first block is

$$|0\rangle \otimes |\psi_k\rangle \xrightarrow[\text{block}]{\text{first}} \frac{1}{\sqrt{2^m}} \sum_{\ell=0}^{2^m-1} \mathrm{e}^{\frac{2\pi i k\ell}{r}} |\ell\rangle|\psi_k\rangle. \tag{10.3}$$

We cannot prepare $|\psi_k\rangle$ as an input to the phase estimation algorithm but we can find a known vector that is spanned by the set of vectors $\{|\psi_0\rangle, ...|\psi_r\rangle\}$. Using the inverse Fourier transform, we have

$$\left|a^\ell\right\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} \mathrm{e}^{\frac{2\pi i k\ell}{r}} |\psi_k\rangle.$$

The simplest choice is $\left|a^0\right\rangle = |1\rangle$, which is given by

$$|1\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} |\psi_k\rangle. \tag{10.4}$$

Using transformation (10.3) for each k, the output of the first block is

$$|0\rangle \otimes |1\rangle \xrightarrow[\text{block}]{\text{first}} \frac{1}{\sqrt{r2^m}} \sum_{k=0}^{r-1} \sum_{\ell=0}^{2^m-1} \mathrm{e}^{\frac{2\pi i k\ell}{r}} |\ell\rangle|\psi_k\rangle.$$

To simplify the output, we use Eq. (10.2)

$$\text{output} = \frac{1}{\sqrt{r2^m}} \sum_{k=0}^{r-1} \sum_{\ell=0}^{2^m-1} \mathrm{e}^{\frac{2\pi i k\ell}{r}} |\ell\rangle \left(\frac{1}{\sqrt{r}} \sum_{\ell'=0}^{r-1} \mathrm{e}^{-\frac{2\pi i k\ell'}{r}} \left|a^{\ell'}\right\rangle \right).$$

Inverting the order of the sums and combining the exponents, we obtain

$$\text{output} = \frac{1}{\sqrt{2^m}} \sum_{\ell=0}^{2^m-1} \sum_{\ell'=0}^{r-1} \left(\frac{1}{r} \sum_{k=0}^{r-1} \mathrm{e}^{\frac{2\pi i k(\ell-\ell')}{r}} \right) |\ell\rangle \left|a^{\ell'}\right\rangle.$$

The expression inside the parenthesis is 1 if $\ell = \ell'$ and 0 otherwise. This means that the output of the first block is

$$|0\rangle^{\otimes m} \otimes |0\rangle^{\otimes(n-1)}|1\rangle \xrightarrow[\text{block}]{\text{first}} \frac{1}{\sqrt{2^m}} \sum_{\ell=0}^{2^m-1} |\ell\rangle \left|a^\ell\right\rangle.$$

This is the same state as in the standard Shor's factoring algorithm (state $|\psi_2\rangle$ of Sec. 7.6) just before applying the inverse Fourier transform F_{2^m}, that is, we consider the part of the algorithm where the input is $|0\rangle^{\otimes m}|0\rangle$ and then the Hadamard gate is applied on each qubit of the first register and then U_f, where $f(x) = a^x \mod N$:

$$|0\rangle^{\otimes m} \otimes |0\rangle^{\otimes n} \xrightarrow{U_f \cdot (H^{\otimes m} \otimes I)} \frac{1}{\sqrt{2^m}} \sum_{\ell=0}^{2^m-1} |\ell\rangle \left|a^\ell\right\rangle.$$

This means that the phase estimation version yields the same result and the analysis of the success probability is exactly the same as in Shor's factoring algorithm if we choose m so that $m = \lceil 2 \log_2 N \rceil$. The number of qubits of the first register must be close to twice the number of qubits of the second register. Fig. 10.3 depicts the quantum part of Shor's algorithm using the quantum phase estimation, where U is given by Eq. (10.1) and a is picked uniformly at random in $\mathbb{Z}_N^*$. Note that the input $|1\rangle$ to the second register is the second vector of the computational basis. This circuit replaces Algorithm 7.2 of Sec. 7.5.

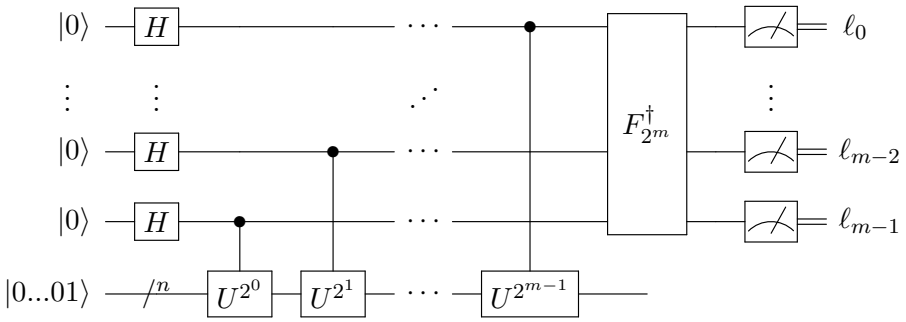
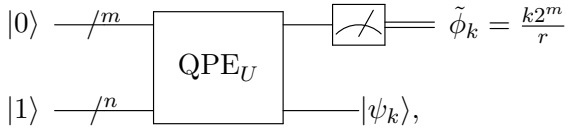


Figure 10.3: Quantum part of Shor's algorithm based on QPE. U is given by Eq. (10.1). The output is the same as the one described in Algorithm 7.2 of Sec. 7.5.

There is an interesting special case. If we know somehow that the order r is a power of 2, we can take $m = n = \lceil \log_2 N \rceil$. If r is a power of 2, the phase of the eigenvalue $\exp(2\pi \mathrm{i} k/r)$ is a rational multiple of 2π, and the phase estimation algorithm returns an exact value $k2^m/r$. In this case, we do not need to calculate the continued fraction expansion of the result. Instead, we simply divide the output by 2^m and select the denominator as the candidate for the order of a.

It is simpler to check that Shor's factoring algorithm works correctly when we use Kitaev's version. Let's suppose that the order r is a power of 2. After compacting the circuit of Fig. 10.3, the output of the QPE algorithm when the input is $|0\rangle|1\rangle$ (in the decimal notation) is shown below



where $0 \leq k < r$ is selected uniformly at random. The result of the last circuit is obtained by using Eq. (10.4) and the fact that the phase of the eigenvalue associated with $|\psi_k\rangle$ is $\phi_k = k/r$. After applying the QPE algorithm, we obtain

$$|0\rangle|1\rangle \xrightarrow{\mathrm{QPE}_U} \frac{1}{\sqrt{r}} \sum_{k=1}^{r-1} |\tilde{\phi}_k\rangle|\psi_k\rangle,$$

where $\tilde{\phi}_k = \frac{k2^m}{r}$. Then, after a measurement of the first register, the output is $\tilde{\phi}_k$ with probability 1 for some $0 \leq k < 1$ uniformly at random, and the outcome of the second register is necessarily the eigenvector $|\psi_k\rangle$. If the order r is not a power of 2, the output $\tilde{\phi}_k$ is a good approximation

for $\frac{k2^m}{r}$, and it is likely that $\tilde{\phi}_k$ be a nearest integer to a multiple of k/r. A candidate r' for the multiplicative order of a modulo N is obtained by selecting the convergent of the continued fraction expansion of $\tilde{\phi}_k/2^m$ that has the largest denominator r' such that $r' < N$. The lower bound for the success probability of the quantum part determined in Sec. 7.6 is valid here.

How do we implement U^{2^j} efficiently for $0 \leq j < m$? Ref. [38] addresses this question. Note that

$$U^{2^j}|y\rangle = \left|a^{2^j} y \mod N\right\rangle.$$

Since a^{2^j} can be calculated efficiently in $O(n^2)$ steps using the repeated squaring method, instead of applying U repeatedly 2^j times, for each j we implement an operator $U_j|y\rangle = |zy\rangle$ after calculating $z = a^{2^j}$ using the repeated squaring method. In this case, the first block can be computed in $O(n^3)$ steps.

10.3 Application to discrete logarithm

Let N, a, and b be known positive integers and let s be a positive integer such that $a^s \equiv b \mod N$ and $\gcd(a, N) = 1$. Our goal is to find s given N, a, and b as input. This is the same problem addressed in Sec. 8.3 on Page 78. Now we show how to solve the discrete logarithm problem using the phase estimation algorithm providing an alternative version to Shor's algorithm for discrete logarithm.

The strategy we use here is the same one used in the order-finding algorithm based on the phase estimation. Recall that, when we described the order-finding algorithm, the output of the first block is the same as in the original Shor's factoring algorithm right before the action of the inverse Fourier transform. Now, the state of the qubits right before the action of $F_r^\dagger \otimes F_r^\dagger$ in Shor's discrete logarithm algorithm described in Sec. 8.3 is

$$\frac{1}{r}\sum_{x,y=0}^{r-1} |x\rangle|y\rangle|a^x b^y \mod N\rangle.$$

If we wish to produce this state using the phase estimation algorithm, we need to use three registers and two unitary operators:

$$\begin{aligned} U_a|x\rangle &= |ax \mod N\rangle, \\ U_b|y\rangle &= |by \mod N\rangle. \end{aligned}$$

U_b is a unitary operator because $\gcd(b, N) = 1$. Indeed, the inverse of b is a^{r-s}, where r is the order of a modulo N. In the new algorithm, the action of U_a is controlled by the first register, and the action of U_b is controlled by the second register, as described in Fig. 10.4. Note that U_a and U_b act on the same register.

Let us check that the vectors

$$|\psi_k\rangle = \frac{1}{\sqrt{r}}\sum_{\ell=0}^{r-1} \mathrm{e}^{-\frac{2\pi \mathrm{i} k\ell}{r}}\left|a^\ell\right\rangle,$$

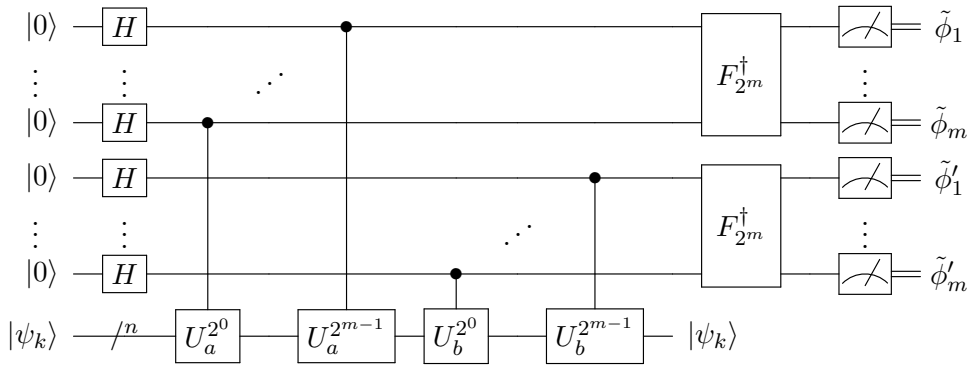


Figure 10.4: Circuit of the discrete logarithm algorithm based on QPE, where $|\psi_k\rangle$ is a common eigenvector of U_a and U_b. Since we are not usually able to prepare $|\psi_k\rangle$ as the input to the third register, we use $|1\rangle$ in its place, which can be represented as a linear combination of $|\psi_k\rangle$, for $0 \leq k < r$.

which are eigenvectors of U_a, are also eigenvectors of U_b. In fact, using that $b = a^s$, we have

$$\begin{aligned} U_b|\psi_k\rangle &= \frac{1}{\sqrt{r}} \sum_{\ell=0}^{r-1} \mathrm{e}^{-\frac{2\pi \mathrm{i} k\ell}{r}} \left|a^{\ell+s}\right\rangle \\ &= \frac{1}{\sqrt{r}} \sum_{\ell=0}^{r-1} \mathrm{e}^{-\frac{2\pi \mathrm{i} k(\ell-s)}{r}} \left|a^{\ell}\right\rangle \\ &= \mathrm{e}^{\frac{2\pi \mathrm{i} ks}{r}} |\psi_k\rangle. \end{aligned}$$

We conclude that $|\psi_k\rangle$ is an eigenvector of U_b with eigenvalue $\exp(2\pi \mathrm{i} ks/r)$ for $0 \leq k < r$. Better yet, $|\psi_k\rangle$ is an eigenvector of U_a and U_b simultaneously. If we are able to prepare the input to the third register of the circuit depicted in Fig. 10.4 as $|\psi_k\rangle$ for some k, we will obtain an estimate of $\tilde{\phi} \approx k/r$ as the output of the first register and an estimate of $\tilde{\phi}' \approx ks/r$ as the output of the second register modulo N. If the input to the third register is $|\psi_k\rangle$, the output of the first block is

$$|0\rangle^{\otimes m}|0\rangle^{\otimes m}|\psi_k\rangle \xrightarrow[\text{block}]{\text{first}} \left(\frac{1}{\sqrt{2^m}} \sum_{\ell=0}^{2^m-1} \mathrm{e}^{\frac{2\pi \mathrm{i} k\ell}{r}} |\ell\rangle\right) \left(\frac{1}{\sqrt{2^m}} \sum_{\ell'=0}^{2^m-1} \mathrm{e}^{\frac{2\pi \mathrm{i} ks\ell'}{r}} \left|\ell'\right\rangle\right) |\psi_k\rangle,$$

where m is the number of qubits of the first and second registers, and $n = \lceil \log_2 N \rceil$ of the third register. The first term inside the parentheses is obtained by replacing ϕ with k/r and the second term by replacing ϕ with ks/r in the output of the first block of the phase estimation algorithm. Simplifying the output, we have

$$|0\rangle^{\otimes m}|0\rangle^{\otimes m}|\psi_k\rangle \xrightarrow[\text{block}]{\text{first}} \left(\frac{1}{2^m} \sum_{\ell,\ell'=0}^{2^m-1} \mathrm{e}^{\frac{2\pi \mathrm{i} k(\ell+s\ell')}{r}} |\ell\rangle\left|\ell'\right\rangle\right) |\psi_k\rangle.$$

Usually, we are not able to prepare $|\psi_k\rangle$ as an input to the phase estimation algorithm but

we can use

$$|1\rangle \;=\; \frac{1}{\sqrt{r}}\sum_{k=0}^{r-1}|\psi_k\rangle$$

again instead of $|\psi_k\rangle$. In this case, the input to and output of the first block are

$$|0\rangle^{\otimes m}|0\rangle^{\otimes m}|1\rangle \xrightarrow[\text{block}]{\text{first}} \frac{1}{\sqrt{r}}\sum_{k=0}^{r-1}\left(\frac{1}{2^m}\sum_{\ell,\ell'=0}^{2^m-1}\mathrm{e}^{\frac{2\pi\mathrm{i}k(\ell+s\ell')}{r}}|\ell\rangle\big|\ell'\big\rangle\right)|\psi_k\rangle,$$

where the input $|1\rangle$ is written in the decimal notation. We use the definition of $|\psi_k\rangle$ given by Eq. (10.2) to simplify the output. We start writing

$$\frac{1}{\sqrt{r}}\sum_{k=0}^{r-1}\left(\frac{1}{2^m}\sum_{\ell,\ell'=0}^{2^m-1}\mathrm{e}^{\frac{2\pi\mathrm{i}k(\ell+s\ell')}{r}}|\ell\rangle\big|\ell'\big\rangle\right)\left(\frac{1}{\sqrt{r}}\sum_{k'=0}^{r-1}\mathrm{e}^{-\frac{2\pi\mathrm{i}kk'}{r}}\Big|a^{k'}\Big\rangle\right),$$

and by pushing the sums over ℓ, ℓ', k' to the left and combining the exponents, we obtain

$$\frac{1}{2^m}\sum_{\ell,\ell'=0}^{2^m-1}\sum_{k'=0}^{r-1}\left(\frac{1}{r}\sum_{k=0}^{r-1}\mathrm{e}^{\frac{2\pi\mathrm{i}k(\ell+s\ell'-k')}{r}}\right)|\ell\rangle\big|\ell'\big\rangle\Big|a^{k'}\Big\rangle.$$

The expression inside the parenthesis is 1 if $\ell+s\ell'=k'$ and 0 otherwise. This means that the action of the first block is

$$|0\rangle^{\otimes m}|0\rangle^{\otimes m}|1\rangle \xrightarrow[\text{block}]{\text{first}} \frac{1}{2^m}\sum_{\ell,\ell'=0}^{2^m-1}|\ell\rangle\big|\ell'\big\rangle\Big|a^{\ell+s\ell'}\Big\rangle.$$

Using that $a^s = b \mod N$, $\ell\to x$, and $\ell'\to y$, we have

$$|0\rangle^{\otimes m}|0\rangle^{\otimes m}|1\rangle \xrightarrow[\text{block}]{\text{first}} \frac{1}{2^m}\sum_{x,y=0}^{2^m-1}|x\rangle|y\rangle|a^x b^y\rangle.$$

The output of the first block is the same state as in the standard Shor's discrete-logarithm algorithm just before applying the inverse Fourier transforms $F_{2^m}\otimes F_{2^m}$ (state $|\psi_2\rangle$ of Algorithm 8.1 on Page 79). This means that the phase estimation version yields the same result and the analysis of the success probability is exactly the same as in Shor's algorithm if we choose m appropriately.

10.4 Application to quantum counting

In the context of Grover's algorithm, we have an oracle $f:\{0,...,N-1\}\to\{0,1\}$ that is a Boolean function defined as

$$f(x)=\begin{cases}1, & \text{if } x\in M,\\ 0, & \text{otherwise,}\end{cases}$$

where M is a subset of the domain and $N=2^n$. We say that x is marked if $x\in M$. The optimal number of steps of Grover's algorithm depends on $|M|$; indeed, it is given by $\frac{\pi}{4}\sqrt{N/|M|}$. If the

cardinality of M is unknown, it is possible to find a marked element by repeatedly guessing the runtime of Grover's algorithm [10]. An alternative method is by solving the quantum counting problem [11].

The quantum counting problem asks what is the cardinality of M given function f as an oracle. A classical solution cannot perform better than $\Omega(N)$ queries to the oracle because all domain elements must be checked. The quantum algorithm can find the solution in $O\big(\sqrt{|M|\,N}\big)$ queries to the oracle.

Before addressing the quantum counting problem, let us review some key points of Grover's algorithm with many marked elements, which is an extension of the algorithm presented in Chapter 9. There is an economical version of the algorithm, which uses only one n-qubit register. The initial state is the uniform superposition of the computational basis given by

$$|\mathrm{d}\rangle \;=\; \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} |j\rangle,$$

and the algorithm consists of $\frac{\pi}{4}\sqrt{N/|M|}$ applications of the evolution operator

$$U \;=\; G\,U_f,$$

where

$$G \;=\; 2|\mathrm{d}\rangle\langle \mathrm{d}| - I$$

and

$$U_f \;=\; \sum_{x=0}^{N-1} (-1)^{f(x)} |x\rangle\langle x|.$$

The analysis of the algorithm is performed by using the $\mathrm{e}^{\pm \mathrm{i}\theta}$-eigenvectors of U, which are given in terms of the superposition of marked states[1] $|M\rangle$ and the superposition of unmarked states $\big|M^{\perp}\big\rangle$:

$$\big|\psi^{\pm}\big\rangle \;=\; \frac{|M\rangle \pm \mathrm{i}\,\big|M^{\perp}\big\rangle}{\sqrt{2}}, \tag{10.5}$$

where

$$\sin\frac{\theta}{2} \;=\; \sqrt{\frac{|M|}{N}},$$

and

$$\begin{aligned} |M\rangle \;&=\; \frac{1}{\sqrt{|M|}} \sum_{x\in M} |x\rangle, \\ \big|M^{\perp}\big\rangle \;&=\; \frac{1}{\sqrt{N-|M|}} \sum_{x\notin M} |x\rangle. \end{aligned}$$

It is straightforward to check that $\big\langle M^{\perp}\big|M\big\rangle = 0$, and

$$|\mathrm{d}\rangle \;=\; \sqrt{\frac{|M|}{N}}\,|M\rangle + \sqrt{1-\frac{|M|}{N}}\,\big|M^{\perp}\big\rangle.$$

[1] Some references call $|M\rangle$ as "good state" and $\big|M^{\perp}\big\rangle$ as "bad state".

Using the equation above, the definition of $\sin(\theta/2)$, and Eq. (10.5), we obtain

$$|\mathrm{d}\rangle \;=\; \frac{\mathrm{e}^{\frac{\mathrm{i}\theta}{2}}|\psi^+\rangle - \mathrm{e}^{-\frac{\mathrm{i}\theta}{2}}|\psi^-\rangle}{\mathrm{i}\sqrt{2}}.$$

Now we come back to the quantum counting problem using the phase estimation algorithm. Since the eigenvalue of $|\psi^+\rangle$ is $\exp(\mathrm{i}\theta)$, where $\sin(\theta/2) = \sqrt{|M|/N}$, we would obtain an approximation for $|M|$ if we use $|\psi^+\rangle$ as the input to the second register of the phase estimation algorithm with $U = GU_f$. We do not know how to prepare $|\psi^+\rangle$, then the strategy is to replace $|\psi\rangle$ in Algorithm 10.1 by a known vector that belongs to the subspace spanned by $|\psi^+\rangle$ and $|\psi^-\rangle$. The best candidate is $|\mathrm{d}\rangle$, which can be easily prepared by applying $H^{\otimes n}$ to $|0\rangle^{\otimes n}$. In this case, the number of qubits of the second register must be $n = \log_2 N$ and the output of the first block is

$$|0\rangle^{\otimes m}|\mathrm{d}\rangle \xrightarrow[\text{block}]{\text{first}} \frac{\mathrm{e}^{\frac{\mathrm{i}\theta}{2}}}{\mathrm{i}\sqrt{2^{m+1}}}\sum_{\ell=0}^{2^m-1}\mathrm{e}^{2\pi\mathrm{i}\phi^+\ell}|\ell\rangle\big|\psi^+\big\rangle - \frac{\mathrm{e}^{-\frac{\mathrm{i}\theta}{2}}}{\mathrm{i}\sqrt{2^{m+1}}}\sum_{\ell=0}^{2^m-1}\mathrm{e}^{2\pi\mathrm{i}\phi^-\ell}|\ell\rangle\big|\psi^-\big\rangle,$$

where $\phi^+ = \theta/2\pi$ for the first term and $\phi^- = (2\pi-\theta)/2\pi$ for the second term. After applying the inverse Fourier transform $F_{2^m}^\dagger$, we obtain the following output of the full circuit

$$\frac{\mathrm{e}^{\frac{\mathrm{i}\theta}{2}}}{\mathrm{i}\sqrt{2}}\left|2^m\tilde{\phi}^+\right\rangle - \frac{\mathrm{e}^{-\frac{\mathrm{i}\theta}{2}}}{\mathrm{i}\sqrt{2}}\left|2^m\tilde{\phi}^-\right\rangle,$$

where $\tilde{\phi}$ is a m bit estimate of ϕ. After a measurement in the computational basis, we learn an estimate of ϕ^+ or ϕ^- with equal chance. Let $\tilde{\phi}$ be the measurement result. Using that that $\sin(\theta/2) = \sqrt{|M|/N}$, $\phi^+ = \theta/2\pi$, and $\phi^- = (2\pi-\theta)/2\pi$, the estimate of $|M|$ is $N\sin^2(\pi\tilde{\phi})$ because for the first case we obtain an estimate of $|M|$ using $|\widetilde{M}| = N\sin^2(\pi\tilde{\phi}^+)$ and for the second case $|\widetilde{M}| = N\sin^2(\pi - \pi\tilde{\phi}^-) = N\sin^2(\pi\tilde{\phi}^-)$.

How many qubits has the first register? This is the tricky part. Note that m cannot be equal to n because the number of applications of U would be $2^0 + \cdots + 2^{n-1} = 2^n - 1$. Then, the number of queries to f would be $O(N)$. If we choose $m = n/2$, the number of queries to f would be $\sqrt{N}$, but in this case we obtain an estimate $|\widetilde{M}|$ so that

$$\big|\,|\widetilde{M}| - |M|\,\big| \;=\; O\big(\sqrt{|M|}\big). \tag{10.6}$$

This estimate is not good. For instance, suppose that $|M|$ is around $N/2$. If we wish to know the number of marked elements, and we obtain $|\widetilde{M}|$ with an error as big as $O(\sqrt{N})$, we don't have a good result. To understand what is the problem here, which does not arise in the factoring and discrete logarithm algorithms, we have to analyze carefully the range of values of angle θ we are trying to estimate.

For this analysis, let us assume that $0 < |M| \ll N$, or more formally, $|M| = o(\sqrt{N})$. The expression $\sin(\theta/2) = \sqrt{|M|/N}$ can be written asymptotically as

$$\theta = \frac{2\sqrt{|M|}}{\sqrt{N}} + O\left(\frac{|M|}{N}\right).$$

This means that $\theta/2\pi$ represented in terms of binary digits is of the form $0.0\cdots01\cdots$, where the number of 0's before the first 1 is around $n/2 - \log_2(|M|)/2$. If we choose the size of the

first register so that m is less than $n/2 - \log_2(|M|)/2$, it is likely that we obtain a 0 as the output of the counting algorithm, which is wrong. If we choose $m = n/2$, we will obtain around $\log_2(|M|)/2$ correct significant bits of $\theta/2\pi$. This is an imprecise estimate of $|M|$ compatible with Eq. (10.6). In fact, $|M|$ has $\log_2(|M|)$ bits, and we need to know most of the significant bits in order to have a good estimate of $|M|$.

Chapter 11

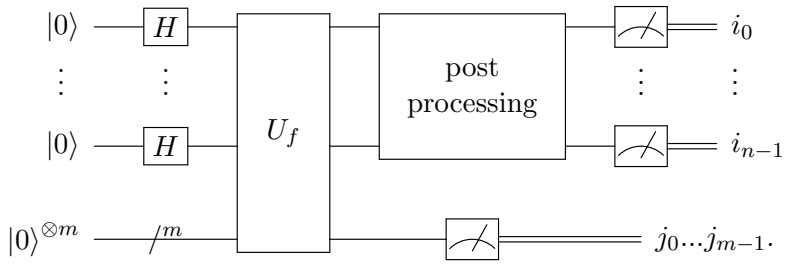
Final Remarks

Most quantum algorithms analyzed in this work can be cast into the oracle-based framework. The query complexity of an algorithm based on an oracle or a black-box is the number of queries. It does not matter how difficult is to implement the oracle unless we aim to solve a practical problem. In practical problems, it is our task to implement the oracle, and then the cost of each evaluation matters. Take Shor's factoring algorithm as an example. The oracle in this case is a r-periodic function and our goal is to find r. We have seen that the function in Shor's algorithm is the modular exponentiation, which can be implemented efficiently in terms of the input size using the repeated squaring method.

Any classical deterministic algorithm is a n-input and m-output function $f : \{0,1\}^n \longrightarrow \{0,1\}^m$, which is a collection of m n-bit Boolean functions. Then, any classical algorithm can be implemented on a quantum computer with two registers of sizes n and m using the operator

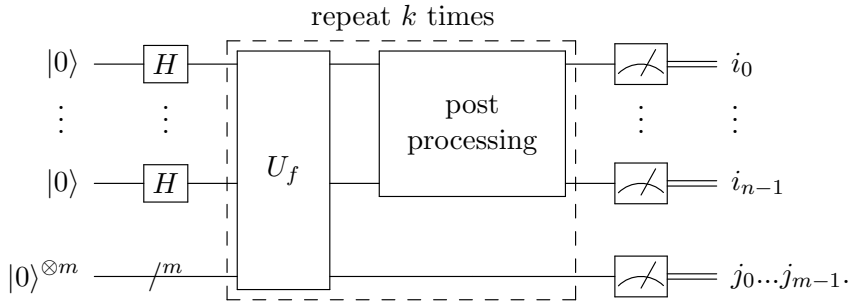
$$U_f|x\rangle|y\rangle \;=\; |x\rangle|y \oplus f(x)\rangle.$$

To exploit quantum parallelism, we have to apply $H^{\otimes n}$ to the first register before applying U_f. After applying U_f, we have a superposition state, which is not useful unless we perform some quantum post-processing that would produce the desired output. Most of the quantum algorithms we have analyzed can be cast into the following circuit:



For the Deutsch-Jozsa, Bernstein-Vazirani, Simon, and Shor (factoring) algorithms, the quantum post-processing is either $H^{\otimes n}$ or the inverse Fourier transform. They have the structure outlined above with a few adaptations. Note that some of those algorithms also require classical post-processing, which is not represented in the quantum circuit.

Grover's algorithm does not have the structure outlined above because U_f and the post-processing are repeated many times before measurement. On the other hand, Grover's has a polynomial gain in contrast with the exponential gain of Simon's and Shor's algorithms. The extension of the general structure that includes Grover's algorithm is

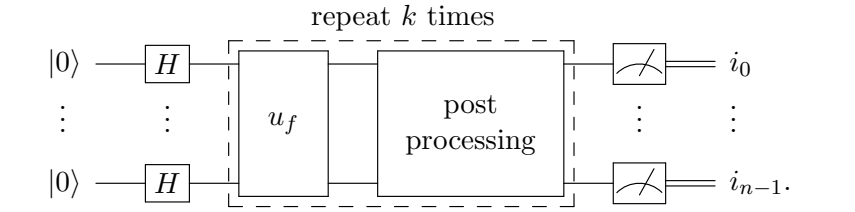


The number of repetitions k is 1 for the Deutsch-Jozsa, Bernstein-Vazirani, Simon, and Shor; and k is $\lfloor \pi\sqrt{2^n}/4 \rfloor$ for Grover's algorithm. The measurement of the second register is unnecessary. It is there because it helps in the analysis of the algorithm.

The second register of the Deutsch-Jozsa, Bernstein-Vazirani, and Grover algorithms has only one qubit ($m = 1$), whose state during the computation is $|-\rangle$, which is obtained by applying X and H on the last qubit before U_f. The oracle for those cases obeys

$$U_f|x\rangle|-\rangle = (-1)^{f(x)}|x\rangle|-\rangle.$$

This means that the second register can be eliminated and there is an economical version of the circuit with the following form:



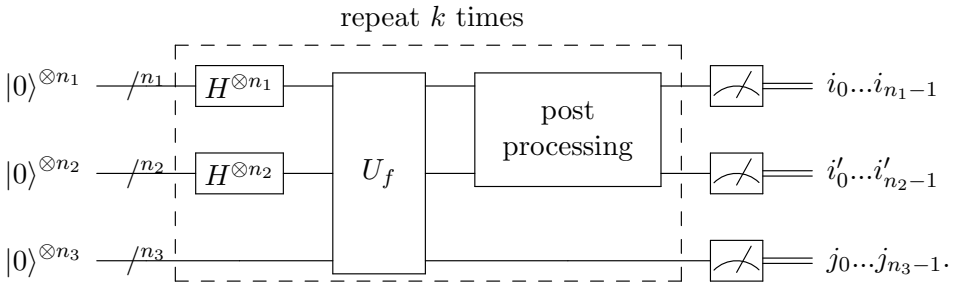
As before, $k = 1$ for the Deutsch-Jozsa and Bernstein-Vazirani algorithms, $k = \lfloor \pi\sqrt{2^n}/4 \rfloor$ for Grover's algorithm, and

$$u_f|x\rangle = (-1)^{f(x)}|x\rangle.$$

Shor's algorithm for discrete logarithms shows us how to extend the structure of the circuit when the function f has more than one variable. Suppose that f has two variables. Then, U_f is defined as

$$U_f|x_1\rangle|x_2\rangle|y\rangle = |x_1\rangle|x_2\rangle|y \oplus f(x_1, x_2)\rangle.$$

This means that we need a circuit with three registers and the general structure of the algorithm is the same up to small changes, as follows:



Those recipes are useful to understand advanced algorithms.

Bibliography

[1] D. Aharonov. Quantum Computation. In *Annual Review of Computational Physics*, pages 259–346. World Scientific, vol. VI, 1999.

[2] S. Axler. *Linear Algebra Done Right*. Springer, New York, 1997.

[3] S. Barnett. *Quantum Information*. Oxford University Press, New York, 2009.

[4] G. Benenti, G. Casati, and G. Strini. *Principles of Quantum Computation and Information: Basic Tools and Special Topics*. World Scientific Publishing, River Edge, 2007.

[5] C. H. Bennett, E. Bernstein, G. Brassard, and U. V. Vazirani. Strengths and weaknesses of quantum computing. *SIAM J. Comput.*, 26(5):1510–1523, 1997.

[6] J. A. Bergou and M. Hillery. *Introduction to the Theory of Quantum Information Processing*. Springer, 2013.

[7] D. J. Bernstein. Detecting perfect powers in essentially linear time. *Math. Comput.*, 67(223):1253–1283, 1998.

[8] E. Bernstein and U. Vazirani. Quantum complexity theory. In *Proc. of the 25th Annual ACM Symposium on Theory of Computing*, STOC '93, page 11–20. ACM, New York, 1993.

[9] E. Bernstein and U. Vazirani. Quantum complexity theory. *SIAM Journal on Computing*, 26(5):1411–1473, 1997.

[10] M. Boyer, G. Brassard, P. Høyer, and A. Tapp. Tight bounds on quantum searching. *Forstschritte Der Physik*, 4:820–831, 1998.

[11] G. Brassard, P. Høyer, M. Mosca, and A. Tapp. Quantum amplitude amplification and estimation. *Quantum Computation and Quantum Information Science, AMS Contemporary Mathematics Series*, 305:53–74, 2002.

[12] G. Brassard, P. Høyer, and A. Tapp. Quantum cryptanalysis of hash and claw-free functions. In *Proc. 3rd Latin American Symposium LATIN'98*, pages 163–169. Springer, 1998.

[13] Guangya Cai and Daowen Qiu. Optimal separation in exact query complexities for Simon's problem. *Journal of Computer and System Sciences*, 97:83–93, 2018.

[14] R. Cleve, A. Ekert, C. Macchiavello, and M. Mosca. Quantum algorithms revisited. *Proc. Royal Society London Ser. A*, 454(1969):339–354, 1998.

[15] D. Coppersmith. An approximate Fourier transform useful in quantum factoring. *Arxiv:quant-ph/0201067*, 2002.

[16] A. DasGupta. The matching, birthday and the strong birthday problem: a contemporary review. *Journal of Statistical Planning and Inference*, 130(1):377–389, 2005.

[17] D. Deutsch. Quantum theory, the Church-Turing principle and the universal quantum computer. *Proc. Royal Society London Ser. A*, pages 96–117, 1985.

[18] D. Deutsch. Quantum computational networks. *Proc. Royal Society London Ser. A*, 425(1868):73–90, 1989.

[19] D. Deutsch and R. Jozsa. Rapid solution of problems by quantum computation. *Proc. Royal Society London Ser. A*, 439(1907):553–558, 1992.

[20] D. Dieks. Communication by EPR devices. *Physics Letters A*, 92(6):271 – 272, 1982.

[21] Jiangfeng Du, Mingjun Shi, Xianyi Zhou, Yangmei Fan, BangJiao Ye, Rongdian Han, and Jihui Wu. Implementation of a quantum algorithm to solve the Bernstein-Vazirani parity problem without entanglement on an ensemble quantum computer. *Phys. Rev. A*, 64:042306, 2001.

[22] M. Ekerå. On the success probability of quantum order finding. *arXiv:2201.07791*, 2022.

[23] L. K. Grover. A fast quantum mechanical algorithm for database search. In *Proc. 28th annual ACM symposium on theory of computing*, STOC '96, pages 212–219, ACM, New York, 1996.

[24] L. K. Grover. Quantum mechanics helps in searching for a needle in a haystack. *Phys. Rev. Lett.*, 79(2):325–328, 1997.

[25] G. H. Hardy and E. M. Wright. *An Introduction to the Theory of Numbers*. Oxford, 4th edition, 1975.

[26] M. Hayashi, S. Ishizaka, A. Kawachi, G. Kimura, and T. Ogawa. *Introduction to Quantum Information Science*. Springer, 2014.

[27] J. Hidary. *Quantum Computing: An Applied Approach*. Springer, 2019.

[28] M. Hirvensalo. *Quantum Computing*. Springer, 2010.

[29] Yan Huang, Zhaofeng Su, Fangguo Zhang, Yong Ding, and Rong Cheng. Quantum algorithm for solving hyperelliptic curve discrete logarithm problem. *Quantum Information Processing*, 19(2):62, 2020.

[30] P. Kaye, R. Laflamme, and M. Mosca. *An Introduction to Quantum Computing*. Oxford University Press, New York, 2007.

[31] A. Yu. Kitaev. Quantum measurements and the Abelian stabilizer problem. *Arxiv:quant-ph/9511026*, 1995.

[32] A. Yu. Kitaev, A. H. Shen, and M. N. Vyalyi. *Classical and Quantum Computation*. American Mathematical Society, Boston, 2002.

[33] C. Lavor, F. Marquezino, A. Oliveira, and R. Portugal. A quantum approach to the discretizable molecular distance geometry problem. *Quantum Information Processing*, 21(7):239, 2022.

[34] R. J. Lipton and K. W. Regan. *Quantum Algorithms via Linear Algebra: A Primer*. MIT Press, 2014.

[35] F. Magniez, M. Santha, and M. Szegedy. Quantum algorithms for the triangle problem. *SIAM J. Comput.*, 37(2):413–424, 2007.

[36] A. Mandviwalla, K. Ohshiro, and B. Ji. Implementing Grover's algorithm on the IBM quantum computers. In *2018 IEEE International Conference on Big Data*, pages 2531–2537, 2018.

[37] D. C. Marinescu and G. M. Marinescu. *Approaching Quantum Computing*. Pearson/Prentice Hall, Michigan, 2005.

[38] I. L. Markov and M. Saeedi. Constant-optimized quantum circuits for modular multiplication and exponentiation. *Quantum Info. Comput.*, 12(5-6):361–394, 2012.

[39] F. L. Marquezino, R. Portugal, and F. D. Sasse. Obtaining the quantum Fourier transform from the classical FFT with QR decomposition. *Journal of Computational and Applied Mathematics*, 235(1):74–81, 2010.

[40] N. D. Mermin. *Quantum Computer Science: An Introduction*. Cambridge University Press, New York, 2007.

[41] D. A. Meyer. Sophisticated quantum search without entanglement. *Phys. Rev. Lett.*, 85:2014–2017, 2000.

[42] Takashi Mihara and Shao Chin Sung. Deterministic polynomial-time quantum algorithms for Simon's problem. *Computational Complexity*, 12(3):162–175, 2003.

[43] A. Montanaro, R. Jozsa, and G. Mitchison. On exact quantum query complexity. *Algorithmica*, 71(4):775–796, 2015.

[44] M. Nakahara and T. Ohmi. *Quantum Computing: From Linear Algebra to Physical Realizations*. CRC Press, 2008.

[45] M. A. Nielsen and I. L. Chuang. *Quantum computation and quantum information*. Cambridge University Press, New York, 2000.

[46] I. Niven, H. S. Zuckerman, and H. L. Montgomery. *An Introduction to the Theory of Numbers*. Wiley, 5th edition, 1991.

[47] J. L. Park. The concept of transition in quantum mechanics. *Foundations of Physics*, 1(1):23–33, 1970.

[48] A. Pavlidis and D. Gizopoulos. Fast quantum modular exponentiation architecture for Shor's factoring algorithm. *Quantum Info. Comput.*, 14(7&8):649–682, 2014.

[49] R. Portugal. *Quantum Walks and Search Algorithms*. Springer, Cham, 2th edition, 2018.

[50] R. Portugal and F. L. Marquezino. Introdução à Programação de Computadores Quânticos. In *Conference CSBC 2019 – 38º JAI*, pages 1–51, Belém, Pará, 2019.

[51] J. Proos and C. Zalka. Shor's discrete logarithm quantum algorithm for elliptic curves. *Quantum Information and Computation*, 3(4):317–344, 2003.

[52] Daowen Qiu and Shenggen Zheng. Generalized Deutsch-Jozsa problem and the optimal quantum algorithm. *Phys. Rev. A*, 97:062331, 2018.

[53] Daowen Qiu and Shenggen Zheng. Revisiting Deutsch-Jozsa algorithm. *Information and Computation*, 275:104605, 2020.

[54] E. Rieffel and W. Polak. *Quantum Computing: a Gentle Introduction*. MIT Press, Cambridge, 2011.

[55] J. B. Rosser and L. Schoenfeld. Approximate formulas for some functions of prime numbers. *Illinois Journal of Mathematics*, 6(1):64 – 94, 1962.

[56] W. Scherer. *Mathematics of Quantum Computing: An Introduction*. Springer, 2019.

[57] P. W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proc. 35th Annual Symposium on Foundations of Computer Science*, pages 124 –134, 1994.

[58] P. W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997.

[59] P. W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Review*, 41(2):303–332, 1999.

[60] D. R. Simon. On the power of quantum computation. In *Proc. 35th Annual Symposium on Foundations of Computer Science*, pages 116–123, 1994.

[61] D. R. Simon. On the power of quantum computation. *SIAM Journal on Computing*, 26(5):1474–1483, 1997.

[62] U. Skosana and M. Tame. Demonstration of Shor's factoring algorithm for $N = 21$ on IBM quantum processors. *Scientific Reports*, 11(1):16599, 2021.

[63] J. Stolze and D. Suter. *Quantum Computing, Revised and Enlarged: A Short Course from Theory to Experiment*. Wiley-VCH, 2008.

[64] G. Strang. *Linear Algebra and Its Applications*. Brooks Cole, 1988.

[65] C. P. Williams. *Explorations in Quantum Computing*. Springer, 2008.

[66] W. K. Wootters and W. H. Zurek. A single quantum cannot be cloned. *Nature*, 299:802–803, 1982.

[67] N. S. Yanofsky and M. Mannucci. *Quantum Computing for Computer Scientists*. Cambridge University Press, 2008.

[68] Zekun Ye, Yunqi Huang, Lvzhou Li, and Yuyi Wang. Query complexity of generalized Simon's problem. *Information and Computation*, 281:104790, 2021.

[69] C. Zalka. Grover's quantum searching algorithm is optimal. *Phys. Rev. A*, 60:2746–2751, 1999.