SIAM Review
Vol. 62, No. 4, pp. 936–981

An Introduction to Quantum Computing, without the Physics*

Giacomo Nannicini†

Abstract. This paper is a gentle but rigorous introduction to quantum computing intended for discrete mathematicians. Starting from a small set of assumptions on the behavior of quantum computing devices, we analyze their main characteristics, stressing the differences with classical computers, and finally describe two well-known algorithms (Simon's algorithm and Grover's algorithm) using the formalism developed in previous sections. This paper does not touch on the physics of the devices, and therefore does not require any notion of quantum mechanics. Numerical examples based on an implementation of Grover's algorithm using open-source software are provided.

Key words. quantum computing, algorithms, open-source software

AMS subject classifications. 68Q12, 81P68

DOI. 10.1137/18M1170650

Contents

*Received by the editors February 13, 2018; accepted for publication (in revised form) October 31, 2019; published electronically November 3, 2020.
https://doi.org/10.1137/18M1170650
Funding: This work was partially supported by the IBM Research Frontiers Institute.
†IBM T.J. Watson, Yorktown Heights, NY 10598 (nannicini@us.ibm.com).

1. Introduction. Quantum computing is a relatively new area of computing that has the potential to greatly speed up the solution of certain problems. However, quantum computers work in a fundamentally different way than classical computers. This introduction aims to explain the basic principles underpinning quantum computing. It assumes the reader is at ease with linear algebra and with basic concepts in classical computing such as Turing machines and algorithm complexity.

The literature contains many textbooks on quantum computing: a comprehensive reference is [30], whereas more modern textbooks that aim to be more accessible to nonphysicists are [29, 33]. However, those books are time-consuming reads and there are not many short introductions that are truly accessible to nonphysicists; [32] is noteworthy, as it actually uses very little physics.

The approach used in this paper is, as far as we are aware, different from the literature in the sense that it abstracts *entirely* away from quantum physics: we study a quantum computing device starting from a small set of assumptions and rigorously derive the remaining properties, focusing on the concepts that are necessary to discuss quantum algorithms. The assumptions are verified in the real world because of the laws of quantum mechanics, but it is not necessary to understand why they hold: as long as we are willing to take a small leap of faith and believe that these assumptions are true, the rest will follow. The exposition in this paper is more formal than in other surveys in the literature, but in some sense it is more mathematically precise: it defines the necessary concepts in a rigorous way, rather than relying on examples or intuition, and provides formal proofs. For this reason, this material is especially suitable for students and researchers in various branches of applied mathematics who will be familiar with the (as far as possible) deductive structure of this paper.

It is important to emphasize that the notation used in this paper is often nonstandard: our choices are meant to facilitate understanding for people who are just learning the basics of the field, and therefore we are mainly concerned with clarity rather than eliminating redundancy. However, in a short paragraph at the end of the paper we highlight some of the major differences between our notation and what is typically found in the literature.

1.1. Overview. The paper is structured as follows.

- In the rest of this section we discuss notation and linear algebra preliminaries.
- In section 2 we define the state of a quantum computer.
- In section 3 we discuss the operations that can be applied by a quantum computer.

- In section 4 we analyze Simon's algorithm, which gives an example of a fundamental principle in quantum algorithms known as *destructive interference*.
- In section 5 we analyze Grover's algorithm, showcasing *amplitude amplification*, another fundamental principle in quantum algorithms.
- Section 6 shows how to implement Grover's algorithm using Qiskit, an open-source Python library for quantum computation.
- Finally, section 7 contains notes for further reading.

The material in this paper is developed to support a graduate-level module in quantum computing. Based on our experience with blackboard-style delivery in the classroom, the material can be split into four modules of 90–120 minutes each, covering sections 1–2, section 3, section 4, and section 5, respectively; plus, if desired, a hands-on class on numerics using section 6, which usually requires 90–120 minutes as well. In the classroom, we suggest introducing Definitions 1.5–1.7 only when they are used, and of course many of the details can be skipped, adjusting the flow as necessary: we highlight with light gray background parts of the material that can be skipped, or briefly summarized, without significantly impairing understanding of subsequent parts.

1.2. Model of Computation. The quantum computing device is, in abstract terms, similar to a classical computing device: it has a state, and that state evolves by applying certain operations. The model of computation that we consider is the quantum circuit model, which works as follows:

1. The quantum computer has a *state* that is contained in a quantum register and is initialized in a predefined way.
2. The state evolves by applying *operations* specified in advance in the form of an algorithm.
3. At the end of the computation, some information on the state of the quantum register is obtained by means of a special operation, called a *measurement*.

All terms in italics will be the subject of the assumptions mentioned earlier, upon which our exposition will be built. Note that this type of computing device is similar to a Turing machine, except for the presence of a tape. It is possible to assume the presence of a tape and be more formal in defining a device that is the quantum equivalent of a Turing machine, but there is no need to do so for the purposes of this work; fundamental results regarding universal quantum computers (i.e., the quantum equivalent of a universal Turing machine) are presented in [16, 37, 9].

We will use the quantum circuit model throughout this paper, including in the numerical example of section 6. This model of computation closely matches that of certain quantum hardware technologies that are used by some of the major players in the field [12], although we should note that the hardware is affected by noise and therefore it does not provide an exact implementation of the theoretical model. To understand the effect of noise, we can give the following simple, but overall quite accurate, intuitive explanation. According to the model of computation, the state evolves by applying operations, and some information on the state can be extracted via a measurement; due to noise, the state may not evolve in the desired way (e.g., applying a certain operation on the state s_1 should yield the state s_2, but we obtain a different state s_3 instead), or the information extracted by a measurement may not be what it is supposed to be (e.g., a measurement should produce the output 0 with probability p_1, but it produces 0 with a different probability p_2 instead).

Since this paper aims to be "physics-free," we will no longer discuss the specifics of existing quantum hardware that follows the circuit model. However, we should note that a different model for quantum computing exists, the so-called adiabatic model.

We do not discuss the adiabatic model for two reasons: first, the adiabatic and the circuit models are equivalent [3], and therefore we are free to choose whatever model is more convenient; second, the circuit model is more natural for computer scientists and is the one used in most textbooks on quantum computing.

1.3. Basic Definitions and Notation. A discussion on quantum computers requires working with the decimal and binary representations of integers, some bit operations, and familiarity with the properties of the tensor product. We describe here the necessary concepts and the notation, so that the reader can come back to this section at any time to clarify symbols.

DEFINITION 1.1. *Given two vector spaces V and W over a field K with bases $e_1, \dots, e_m$ and $f_1, \dots, f_n$, respectively, the* tensor product $V \otimes W$ *is another vector space over K of dimension mn. The tensor product space is equipped with a bilinear operation $\otimes : V \times W \to V \otimes W$. The vector space $V \otimes W$ has basis $e_i \otimes f_j$ $\forall i = 1, \dots, m, j = 1, \dots, n$.*

If the origin vector spaces are complex Euclidean spaces of the form $\mathbb{C}^n$, and we choose the standard basis (consisting of the orthonormal vectors that have a 1 in a single position and 0 elsewhere) in the origin vector spaces, then the tensor product is none other than the Kronecker product, which is itself a generalization of the outer product. This is formalized next.

DEFINITION 1.2. *Given $A \in \mathbb{C}^{m \times n}, B \in \mathbb{C}^{p \times q}$, the* Kronecker product $A \otimes B$ *is the matrix $D \in \mathbb{C}^{mp \times nq}$ defined as*

$$D := A \otimes B = \begin{pmatrix} a_{11}B & \dots & a_{1n}B \\ a_{21}B & \dots & a_{2n}B \\ \vdots & & \vdots \\ a_{m1}B & \dots & a_{mn}B \end{pmatrix}.$$

If we choose the standard basis over the vector spaces $\mathbb{C}^{m \times n}$ and $\mathbb{C}^{p \times q}$, then the bilinear operation $\otimes$ of the tensor product $\mathbb{C}^{m \times n} \otimes \mathbb{C}^{p \times q}$ is simply the Kronecker product.

In this paper we always work with complex Euclidean spaces of the form $\mathbb{C}^n$, using the standard basis. With a slight but common abuse of notation, we will therefore use the tensor product to refer to the Kronecker and outer products.

Example 1.3. We provide an example of the tensor product for normalized vectors, which will link this concept to probability distributions and will hopefully provide a better understanding of some of the future material. Consider two independent discrete random variables X and Y that describe the probability of extracting numbers from two urns. The first urn contains the numbers 0 and 1; the second urn contains the numbers $00, 01, 10, 11$. Assume that the extraction mechanism is biased and therefore the outcomes do not have equal probability. The outcome probabilities are given below, and for convenience we define two vectors containing them:

$$x = \begin{pmatrix} \Pr(X = 0) \\ \Pr(X = 1) \end{pmatrix} = \begin{pmatrix} 0.25 \\ 0.75 \end{pmatrix}, \qquad y = \begin{pmatrix} \Pr(Y = 00) \\ \Pr(Y = 01) \\ \Pr(Y = 10) \\ \Pr(Y = 11) \end{pmatrix} = \begin{pmatrix} 0.2 \\ 0.2 \\ 0.2 \\ 0.4 \end{pmatrix}.$$

Notice that because each vector contains probabilities for all possible respective outcomes, the vectors are normalized so that their entries sum up to 1. Then the joint

probabilities for simultaneously extracting numbers from the two urns are given by the tensor product $x \otimes y$,

$$x \otimes y = \begin{pmatrix} 0.25 \\ 0.75 \end{pmatrix} \otimes \begin{pmatrix} 0.2 \\ 0.2 \\ 0.2 \\ 0.4 \end{pmatrix} = \begin{pmatrix} 0.05 \\ 0.05 \\ 0.05 \\ 0.1 \\ 0.15 \\ 0.15 \\ 0.15 \\ 0.3 \end{pmatrix} = \begin{pmatrix} \Pr(X=0)\Pr(Y=00) \\ \Pr(X=0)\Pr(Y=01) \\ \Pr(X=0)\Pr(Y=10) \\ \Pr(X=0)\Pr(Y=11) \\ \Pr(X=1)\Pr(Y=00) \\ \Pr(X=1)\Pr(Y=01) \\ \Pr(X=1)\Pr(Y=10) \\ \Pr(X=1)\Pr(Y=11) \end{pmatrix} = \begin{pmatrix} \Pr(X=0,Y=00) \\ \Pr(X=0,Y=01) \\ \Pr(X=0,Y=10) \\ \Pr(X=0,Y=11) \\ \Pr(X=1,Y=00) \\ \Pr(X=1,Y=01) \\ \Pr(X=1,Y=10) \\ \Pr(X=1,Y=11) \end{pmatrix},$$

where the last equality is due to the fact that X and Y are independent. The vector $x \otimes y$ is also normalized, which is easy to verify algebraically.

The next proposition states some properties of the tensor product that will be useful in the rest of this paper.

PROPOSITION 1.4. *Let $A, B : \mathbb{C}^{m \times m}, C, D \in \mathbb{C}^{n \times n}$ be linear transformations on V and W, respectively, $u, v \in \mathbb{C}^m, w, x \in \mathbb{C}^n$, and $a, b \in \mathbb{C}$. The tensor product satisfies the following properties:*

(i) $(A \otimes C)(B \otimes D) = AB \otimes CD$.
(ii) $(A \otimes C)(u \otimes w) = Au \otimes Cw$.
(iii) $(u + v) \otimes w = u \otimes w + v \otimes w$.
(iv) $u \otimes (w + x) = u \otimes w + u \otimes x$.
(v) $(au) \otimes (bw) = ab(u \otimes w)$.
(vi) $(A \otimes C)^* = A^* \otimes C^*$.

Above and in what follows, the notation A^* denotes the conjugate transpose of A, which is the matrix defined as $A^* := \bar{A}^\top$. Given a matrix A, the notation $A^{\otimes n}$ indicates the tensor product of A with itself n times, and the same notation will be used for vector spaces $\mathbb{S}$:

$$A^{\otimes n} := \underbrace{A \otimes A \otimes \cdots \otimes A}_{n \text{ times}}, \qquad \mathbb{S}^{\otimes n} := \underbrace{\mathbb{S} \otimes \mathbb{S} \otimes \cdots \otimes \mathbb{S}}_{n \text{ times}}.$$

The quantum computing literature refers to a Hilbert space, typically denoted $\mathcal{H}$, rather than a complex Euclidean space $\mathbb{C}^n$. However, the material discussed in this paper does not require any property of Hilbert spaces that is not already present in complex Euclidean spaces, hence we stick to the more familiar concept.

We will work extensively with binary strings, using the following definitions.

DEFINITION 1.5. *For any integer $q > 0$, we denote by $\vec{\jmath} \in \{0,1\}^q$ a binary string on q digits, where we use the arrow to emphasize that $\vec{\jmath}$ is a string of binary digits rather than an integer. Given $\vec{\jmath} \in \{0,1\}^q$, we denote its kth digit by $\vec{\jmath}_k$.*

We use the notation $\vec{0}$ to denote the all-zero binary string, and $\vec{1}$ to denote the all-one binary string; the size of these strings will always be clear from the context. We use a little-endian convention for binary strings, i.e., the first digit is the most significant one. Thus, the binary string $\vec{\jmath} \in \{0,1\}^q$ corresponds to the decimal number $\sum_{k=1}^{q} \vec{\jmath}_k 2^{q-k}$.

In the rest of this paper, as is common in the quantum computing literature, we use $\vec{\jmath} \in \{0,1\}^q$ to index the elements of 2^q-dimensional vectors; such an index is well-defined because $\{0,1\}^q$ has 2^q elements.

DEFINITION 1.6. *For any integer $q > 0$ and binary strings $\vec{j}, \vec{k} \in \{0,1\}^q$, we denote by $\vec{j} \oplus \vec{k}$ the bitwise modulo 2 addition of q-digit strings (bitwise XOR), defined as*

$$\vec{j} \oplus \vec{k} = \vec{h} \text{ with } \vec{h} \in \{0,1\}^q \text{ and } \vec{h}_p = \begin{cases} 0 & \text{if } \vec{j}_p = \vec{k}_p, \\ 1 & \text{otherwise,} \end{cases} \forall p = 1, \ldots, q.$$

DEFINITION 1.7. *For any integer $q > 0$ and binary strings $\vec{j}, \vec{k} \in \{0,1\}^q$, we denote by $\vec{j} \bullet \vec{k}$ the bitwise dot product of q-digit strings, defined as*

$$\vec{j} \bullet \vec{k} = \sum_{h=1}^{q} \vec{j}_h \vec{k}_h.$$

The last piece of notation that we need is the *bra-ket* notation, used in quantum mechanics. As mentioned earlier, this paper will not touch on any quantum-mechanical concepts; however, there is an undeniable advantage in the quantum notation in that it puts the most important information in the center of the symbols, rather than relegate it to a marginal role in the subscript or superscript. Furthermore, a goal of this work is to equip the reader with the necessary tools to understand quantum computing papers, and hence it is important to explain the bra-ket notation.

DEFINITION 1.8. *Given a complex Euclidean space $\mathbb{S} \equiv \mathbb{C}^n$, $|\psi\rangle \in \mathbb{S}$ denotes a column vector, and $\langle\psi| \in \mathbb{S}^*$ denotes a row vector that is the conjugate transpose of $|\psi\rangle$, i.e., $\langle\psi| = |\psi\rangle^*$. The vector $|\psi\rangle$ is also called a* ket*, and the vector $\langle\psi|$ is also called a* bra*.*

Thus, an expression such as $\langle\psi|\phi\rangle$ is an inner product. The complex Euclidean spaces used in this work will be of the form $(\mathbb{C}^2)^{\otimes q}$, where q is a given integer. It is therefore convenient to specify the basis elements of such spaces.

DEFINITION 1.9. *The standard basis for $\mathbb{C}^2$ is denoted by $|0\rangle = \binom{1}{0}, |1\rangle = \binom{0}{1}$. The standard basis for $(\mathbb{C}^2)^{\otimes q}$, which has 2^q elements, is denoted by $|\vec{j}\rangle, \vec{j} \in \{0,1\}^q$.*

According to our notation, for any q-digit binary string $\vec{j} \in \{0,1\}^q$, $|\vec{j}\rangle$ is the 2^q-dimensional basis vector in $(\mathbb{C}^2)^{\otimes q}$ corresponding to the binary string $\vec{j}$. Since we always use the standard basis and the most natural order for its vectors, it is easy to verify that for $\vec{j} \in \{0,1\}^q$, $|\vec{j}\rangle$ is the basis vector with a 1 in position $\sum_{k=1}^{q} \vec{j}_k 2^{q-k} + 1$, and 0 elsewhere. For example, $|101\rangle$ is the 8-dimensional basis vector $(0\,0\,0\,0\,0\,1\,0\,0)^\top$, obtained as the tensor product $|1\rangle \otimes |0\rangle \otimes |1\rangle$. Whenever useful for clarity, we use a subscript for bras and kets to denote the dimension of the space that the vector belongs to, e.g., we write $|\vec{j}\rangle_q$ to emphasize that we are working in a 2^q-dimensional space (or, in other words, that the basis elements of the space are associated with binary strings with q digits). We typically omit the subscript if the dimension of the space is evident from the context. We provide a further example of this notation below.

Example 1.10. Let us write the basis elements of $(\mathbb{C}^2)^{\otimes 2} = \mathbb{C}^2 \otimes \mathbb{C}^2$ as

$$|00\rangle_2 = |00\rangle = |0\rangle \otimes |0\rangle = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \qquad |01\rangle_2 = |01\rangle = |0\rangle \otimes |1\rangle = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix},$$

$$|10\rangle_2 = |10\rangle = |1\rangle \otimes |0\rangle = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}, \qquad |11\rangle_2 = |11\rangle = |1\rangle \otimes |1\rangle = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}.$$

In the above example we made an exception to our rule and used a subscript to denote the dimension of the basis vectors, just to emphasize that $|00\rangle_2$ and $|00\rangle$ are exactly the same. In the remainder of this paper, we will write $|01\rangle$ rather than $|01\rangle_2$ because it is clear that the basis element $|01\rangle$ has two digits and therefore lives in the space $(\mathbb{C}^2)^{\otimes 2}$.

To improve clarity when dealing with vectors in $(\mathbb{C}^2)^{\otimes q}$, we always denote basis vectors using spelled-out binary strings or Roman letters (e.g., $|01\rangle, |\vec{j}\rangle, |\vec{h}\rangle, |\vec{x}\rangle, |\vec{y}\rangle$ all denote basis vectors), whereas we use Greek letters to denote vectors that may not be basis vectors (e.g., $|\psi\rangle, |\phi\rangle$ all denote vectors that may not be basis vectors). In the same spirit, single-digit binary numbers are always denoted with Roman letters (e.g., x, y, z denote a 0 or a 1).

2. Qubits and Quantum States. According to our computational model, a quantum computing device has a state that is stored in the quantum register. Qubits are the quantum counterpart of the bits found in classical computers: a classical computer has registers that are made up of bits, whereas a quantum computer has a single quantum register that is made up of qubits. The assumption that there is a single quantum register is without loss of generality, as one can think of multiple registers as being placed "side-by-side" to form a single register (of course, one would then need to specify what operations are allowed on the resulting register). The state of the quantum register, and therefore of the quantum computing device, is defined next.

ASSUMPTION 2.1. *The state of a q-qubit quantum register is a unit vector in*

$$\left(\mathbb{C}^2\right)^{\otimes q} = \underbrace{\mathbb{C}^2 \otimes \cdots \otimes \mathbb{C}^2}_{q \text{ times}}.$$

Remark 2.2. A vector $|\psi\rangle \in \mathbb{C}^n$ is a unit vector if $\||\psi\rangle\| = \sqrt{\langle\psi|\psi\rangle} = 1$.

Remark 2.3. Choosing the standard basis for $\mathbb{C}^2$, the state of a 1-qubit register ($q = 1$) can be represented as $\alpha|0\rangle + \beta|1\rangle = \alpha\binom{1}{0} + \beta\binom{0}{1} = \binom{\alpha}{\beta}$, where $\alpha, \beta \in \mathbb{C}$ and $|\alpha|^2 + |\beta|^2 = 1$.

Remark 2.4. Given the standard basis for $\mathbb{C}^2$, a basis for $\left(\mathbb{C}^2\right)^{\otimes q}$ is given by the following 2^q vectors:

$$\begin{aligned}
|\underbrace{00\cdots00}_{q \text{ digits}}\rangle &= \underbrace{|0\rangle \otimes \cdots \otimes |0\rangle \otimes |0\rangle}_{q \text{ times}}, \\
|\underbrace{00\cdots01}_{q \text{ digits}}\rangle &= \underbrace{|0\rangle \otimes \cdots \otimes |0\rangle \otimes |1\rangle}_{q \text{ times}}, \\
&\vdots \\
|\underbrace{11\cdots11}_{q \text{ digits}}\rangle &= \underbrace{|1\rangle \otimes \cdots \otimes |1\rangle \otimes |1\rangle}_{q \text{ times}}.
\end{aligned}$$

In more compact form, the vectors are denoted by $|\vec{j}\rangle, \vec{j} \in \{0,1\}^q$. The state of a q-

qubit quantum register can then be represented as $|\psi\rangle = \sum_{\vec{j}\in\{0,1\}^q} \alpha_{\vec{j}}|\vec{j}\rangle_q$ with $\alpha_{\vec{j}} \in \mathbb{C}$ and $\sum_{\vec{j}\in\{0,1\}^q} |\alpha_{\vec{j}}|^2 = 1$.

For brevity, we often write "state of q-qubits" or "q-qubit state" to refer to the state of a q-qubit quantum register. This is common in the literature, where the discussion of qubits is not necessarily limited to the context of quantum registers. Given certain properties of the tensor product, we will see that sometimes it is appropriate to refer to the state of just some of the qubits of a quantum computing device, rather than all of them, and that this may still be a well-defined concept. We will revisit this in section 2.2.

It is important to remark that $\left(\mathbb{C}^2\right)^{\otimes q}$ is a 2^q-dimensional space. This is in sharp contrast with the state of classical bits: given q classical bits, their state is a binary string in $\{0,1\}^q$, which is a q-dimensional space. In other words, the dimension of the state space of quantum registers grows *exponentially* in the number of qubits, whereas the dimension of the state space of classical registers grows *linearly* in the number of bits. Furthermore, to represent a quantum state we need complex coefficients: the state of a q-qubit quantum register is described by 2^q complex coefficients, which is an enormous amount of information compared to what is necessary to describe a q-bit classical register. However, we will see in section 3.2 that a quantum state cannot be accessed directly, therefore even if a description of the quantum state requires infinite precision in principle, we cannot access such a description as easily as with classical registers. In fact, as it turns out we cannot extract more than q bits of information from a q-qubit register! This will be intuitively clear after stating the effect of quantum measurements in section 3.2; for a formal proof, see [24].

2.1. Basis States and Superposition. We continue our study of the state of quantum registers by discussing the concept of superposition.

DEFINITION 2.5. *We say that q qubits are in a* basis state *if the state $|\psi\rangle = \sum_{\vec{j}\in\{0,1\}^q} \alpha_{\vec{j}}|\vec{j}\rangle_q$ of the corresponding register is such that $\exists \vec{k} : |\alpha_{\vec{k}}| = 1$, $\alpha_{\vec{j}} = 0\ \forall \vec{j} \neq \vec{k}$. Otherwise, we say that they are in a* superposition.

Remark 2.6. A simpler, more intuitive definition would be to say that a basis state is such that $|\psi\rangle = |\vec{k}\rangle$ for some $\vec{k} \in \{0,1\}^q$. It is acceptable to use the simpler definition if desired: as it turns out, even if the states $\alpha_{\vec{k}}|\vec{k}\rangle$ for some $\vec{k} \in \{0,1\}^q$ and $|\alpha_{\vec{k}}|^2 = 1$ are all different in principle, they are equivalent to $|\vec{k}\rangle$ up to the multiplication factor $\alpha_{\vec{k}}$, which will be seen to be unimportant in Example 3.9.

Example 2.7. Consider two 1-qubit registers and their states $|\psi\rangle, |\phi\rangle$:

$$|\psi\rangle = \alpha_0|0\rangle + \alpha_1|1\rangle,$$
$$|\phi\rangle = \beta_0|0\rangle + \beta_1|1\rangle.$$

If we put these 1-qubit registers side-by-side to form a 2-qubit register, then the 2-qubit register will be in state

$$|\psi\rangle \otimes |\phi\rangle = \alpha_0\beta_0|0\rangle \otimes |0\rangle + \alpha_0\beta_1|0\rangle \otimes |1\rangle + \alpha_1\beta_0|1\rangle \otimes |0\rangle + \alpha_1\beta_1|1\rangle \otimes |1\rangle.$$

If both $|\psi\rangle$ and $|\phi\rangle$ are in a basis state, we have that either α_0 or α_1 is zero, and similarly either β_0 or β_1 is zero, while the nonzero coefficients have modulus one. Thus, only one of the coefficients in the expression of the state of $|\psi\rangle \otimes |\phi\rangle$ is nonzero, and in fact its modulus is one. This implies that if both $|\psi\rangle$ and $|\phi\rangle$ are in a basis

state, $|\psi\rangle \otimes |\phi\rangle$ is in a basis state as well. But now assume that $\alpha_0 = \beta_0 = \alpha_1 = \beta_1 = \frac{1}{\sqrt{2}}$: the qubits $|\psi\rangle$ and $|\phi\rangle$ are in a superposition. Then the state of $|\psi\rangle \otimes |\phi\rangle$ is $\frac{1}{2}|00\rangle + \frac{1}{2}|01\rangle + \frac{1}{2}|10\rangle + \frac{1}{2}|11\rangle$, which is a superposition as well. Notice that the normalization of the coefficients works out, as one can easily check with simple algebra: the tensor product of unit vectors is also a unit vector.

The example clearly generalizes to an arbitrary number of qubits. In fact, the following proposition is trivially true.

PROPOSITION 2.8. *For any q, a q-qubit register is in a basis state if and only if its state can be expressed as the tensor product of q 1-qubit registers, each of which is in a basis state.*

Notice that superposition does not have a classical equivalent: q classical bits are always in a basis state, i.e., a q-bit classical register will always contain exactly one of the 2^q binary strings in $\{0,1\}^q$. Indeed, superposition is one of the main features of quantum computers that differentiates them from classical computers. The second important feature is entanglement, which will be discussed next.

2.2. Product States and Entanglement. We have seen that the state of a q-qubit register is a vector in $\left(\mathbb{C}^2\right)^{\otimes q}$, which is a 2^q-dimensional space. Since this is a tensor product of $\mathbb{C}^2$, i.e., the space in which 1-qubit states live, it is natural to ask whether moving from single qubits to multiple qubits has gained us anything at all. In other words, we want to investigate whether the quantum states that are representable on q qubits are simply the tensor product of q 1-qubit states. We can answer this question by using the definitions given above. The state of q qubits is a unit vector in $\left(\mathbb{C}^2\right)^{\otimes q}$, and it can be written as

$$|\psi\rangle = \sum_{\vec{\jmath}\in\{0,1\}^q} \alpha_{\vec{\jmath}} |\vec{\jmath}\rangle_q, \qquad \sum_{\vec{\jmath}\in\{0,1\}^q} |\alpha_{\vec{\jmath}}|^2 = 1.$$

Now let us consider the tensor product of q 1-qubit states, the kth of which is given by $\beta_{k,0}|0\rangle + \beta_{k,1}|1\rangle$ for $k = 1, \ldots, q$ (the first qubit corresponds to the most significant bit, according to the little-endian convention). Taking the tensor product we obtain the vector

$$\begin{aligned}|\phi\rangle &= (\beta_{1,0}|0\rangle + \beta_{1,1}|1\rangle) \otimes (\beta_{2,0}|0\rangle + \beta_{2,1}|1\rangle) \otimes \cdots \otimes (\beta_{q,0}|0\rangle + \beta_{q,1}|1\rangle) \\ &= \sum_{j_1=0}^{1} \sum_{j_2=0}^{1} \cdots \sum_{j_q=0}^{1} \prod_{k=1}^{q} \beta_{k,j_k} |\underbrace{j_1 j_2 \cdots j_q}_{\substack{\text{taken as a} \\ \text{binary string}}}\rangle = \sum_{\vec{\jmath}\in\{0,1\}^q} \prod_{k=1}^{q} \beta_{k,\vec{\jmath}_k} |\vec{\jmath}\rangle_q, \\ &\text{satisfying } |\beta_{k,0}|^2 + |\beta_{k,1}|^2 = 1 \ \forall k = 1, \ldots, q.\end{aligned}$$

The normalization condition for $|\phi\rangle$ implies the normalization condition of $|\psi\rangle$, but the converse is not true. That is, $|\beta_{k,0}|^2 + |\beta_{k,1}|^2 = 1 \ \forall k = 1, \ldots, q$ implies $\sum_{j_1=0}^{1} \sum_{j_2=0}^{1} \cdots \sum_{j_q=0}^{1} \left|\prod_{k=1}^{q} \beta_{k,j_k}\right|^2 = 1$, but not vice versa. This means that there exist values of $\alpha_{\vec{\jmath}}$ with $\sum_{\vec{\jmath}\in\{0,1\}^q} |\alpha_{\vec{\jmath}}|^2 = 1$ that cannot be expressed as coefficients $\beta_{k,0}, \beta_{k,1}$ (for $k = 1, \ldots, q$) satisfying the conditions for $|\phi\rangle$.

This is easily clarified with an example.

Example 2.9. Consider two 1-qubit states:

$$|\psi\rangle = \alpha_0|0\rangle + \alpha_1|1\rangle,$$
$$|\phi\rangle = \beta_0|0\rangle + \beta_1|1\rangle.$$

Taking the two qubits together in a 2-qubit register, the state of the 2-qubit register is

$$|\psi\rangle \otimes |\phi\rangle = \alpha_0\beta_0|00\rangle + \alpha_0\beta_1|01\rangle + \alpha_1\beta_0|10\rangle + \alpha_1\beta_1|11\rangle, \tag{2.1}$$

with the normalization conditions $|\alpha_0|^2 + |\alpha_1|^2 = 1$ and $|\beta_0|^2 + |\beta_1|^2 = 1$. The general state of a 2-qubit register $|\xi\rangle$ is

$$|\xi\rangle = \gamma_{00}|00\rangle + \gamma_{01}|01\rangle + \gamma_{10}|10\rangle + \gamma_{11}|11\rangle, \tag{2.2}$$

with normalization condition $|\gamma_{00}|^2 + |\gamma_{01}|^2 + |\gamma_{10}|^2 + |\gamma_{11}|^2 = 1$. Comparing (2.1) and (2.2), we determine that $|\xi\rangle$ is of the form $|\psi\rangle \otimes |\phi\rangle$ if and only if it satisfies the relationship

$$\gamma_{00}\gamma_{11} = \gamma_{01}\gamma_{10}. \tag{2.3}$$

Clearly, $|\psi\rangle \otimes |\phi\rangle$ yields coefficients that satisfy this condition. To see the converse, let $\theta_{00}, \theta_{01}, \theta_{10}, \theta_{11}$ be the phases of $\gamma_{00}, \gamma_{01}, \gamma_{10}, \gamma_{11}$. Notice that (2.3) implies

$$|\gamma_{00}|^2|\gamma_{11}|^2 = |\gamma_{01}|^2|\gamma_{10}|^2,$$
$$\theta_{00} + \theta_{11} = \theta_{01} + \theta_{10}.$$

Then we can write

$$\begin{aligned}|\gamma_{00}| &= \sqrt{|\gamma_{00}|^2} = \sqrt{|\gamma_{00}|^2(|\gamma_{00}|^2 + |\gamma_{01}|^2 + |\gamma_{10}|^2 + |\gamma_{11}|^2)} \\ &= \sqrt{|\gamma_{00}|^4 + |\gamma_{00}|^2|\gamma_{01}|^2 + |\gamma_{00}|^2|\gamma_{10}|^2 + |\gamma_{01}|^2|\gamma_{10}|^2} \\ &= \underbrace{\sqrt{|\gamma_{00}|^2 + |\gamma_{01}|^2}}_{|\alpha_0|}\underbrace{\sqrt{|\gamma_{00}|^2 + |\gamma_{10}|^2}}_{|\beta_0|},\end{aligned}$$

and similarly for the other coefficients:

$$|\gamma_{01}| = \underbrace{\sqrt{|\gamma_{00}|^2 + |\gamma_{01}|^2}}_{|\alpha_0|}\underbrace{\sqrt{|\gamma_{01}|^2 + |\gamma_{11}|^2}}_{|\beta_1|},$$
$$|\gamma_{10}| = \underbrace{\sqrt{|\gamma_{10}|^2 + |\gamma_{11}|^2}}_{|\alpha_1|}\underbrace{\sqrt{|\gamma_{00}|^2 + |\gamma_{10}|^2}}_{|\beta_0|},$$
$$|\gamma_{11}| = \underbrace{\sqrt{|\gamma_{10}|^2 + |\gamma_{11}|^2}}_{|\alpha_1|}\underbrace{\sqrt{|\gamma_{01}|^2 + |\gamma_{11}|^2}}_{|\beta_1|}.$$

To fully define the coefficients $\alpha_0, \alpha_1, \beta_0, \beta_1$ we must determine their phases. We can assign

$$\alpha_0 = e^{i\theta_{00}}|\alpha_0|, \qquad \alpha_1 = e^{i\theta_{10}}|\alpha_1|, \qquad \beta_0 = |\beta_0|, \qquad \beta_1 = e^{i(\theta_{01}-\theta_{00})}|\beta_1|. \tag{2.4}$$

It is now easy to verify that the state $|\xi\rangle$ in (2.2) can be expressed as $|\psi\rangle \otimes |\phi\rangle$ in (2.1) with coefficients $\alpha_0, \alpha_1, \beta_0, \beta_1$ as given in (2.4).

The condition in (2.3), to verify whether the coefficients of a 2-qubit state $|\xi\rangle$ can be expressed as a tensor product of two 1-qubit states, can also be written in matrix form, which makes it easier to remember. If we assign the rows of the matrix to the first qubit, and the columns to the second qubit, we can arrange the coefficients γ as follows (notice how the first qubit index has value 0 in the first row and 1 in the second row, and similarly for the second qubit index and the columns):

$$\begin{pmatrix} \gamma_{00} & \gamma_{01} \\ \gamma_{10} & \gamma_{11} \end{pmatrix}.$$

Then $|\xi\rangle$ is a tensor product of two 1-qubit states if and only if this matrix has rank 1. This is equivalent to (2.3).

We formalize the concept of expressing a quantum state as a tensor product of lower-dimensional quantum states as follows.

DEFINITION 2.10. *A quantum state* $|\psi\rangle \in \left(\mathbb{C}^2\right)^{\otimes q}$ *is a* product state *if it can be expressed as a tensor product* $|\psi_1\rangle \otimes \cdots \otimes |\psi_q\rangle$ *of* q 1*-qubit states. Otherwise, it is* entangled.

Notice that a general quantum state $|\psi\rangle$ could be the product of two or more lower-dimensional quantum states, e.g., $|\psi\rangle = |\psi_1\rangle \otimes |\psi_2\rangle$, with $|\psi_1\rangle$ and $|\psi_2\rangle$ being entangled states. In such a situation, $|\psi\rangle$ exhibits some entanglement, but in some sense it can still be "simplified." Generally, according to the definition above, we call a quantum state entangled as long as it cannot be fully decomposed into a tensor product of 1-qubit states. In the case of quantum systems composed of multiple subsystems (rather than just two subsystems as in the example $|\psi\rangle = |\psi_1\rangle \otimes |\psi_2\rangle$), the concept of entanglement as discussed in the literature is not as simple as given in Definition 2.10 (and the rank-1 test discussed at the end of Example 2.9 is not well-defined). However, our simplified definition works in this paper and for most of the literature on quantum algorithms, and therefore we can leave other considerations aside; we refer the reader to [14] as an entry point for a discussion on multipartite entanglement.

Example 2.11. Consider the following 2-qubit state:

$$\frac{1}{2}|00\rangle + \frac{1}{2}|01\rangle + \frac{1}{2}|10\rangle + \frac{1}{2}|11\rangle.$$

This is a product state because it is equal to $\left(\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle\right) \otimes \left(\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle\right)$. By contrast, the 2-qubit state

$$\frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle$$

is an entangled state, because it cannot be expressed as a product of two 1-qubit states.

3. Operations on Qubits. Operations on quantum states must satisfy certain conditions, to ensure that applying an operation does not break the basic properties of the quantum state. The required property is stated below, and we treat it as an assumption.

ASSUMPTION 3.1. *An* operation *applied by a quantum computer with* q *qubits, also called a* gate*, is a unitary matrix in* $\mathbb{C}^{2^q \times 2^q}$.

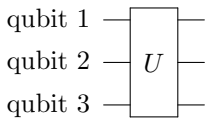


Fig. 1 *A simple quantum circuit.*

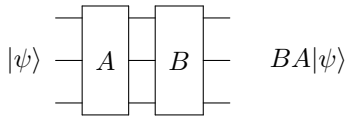


Fig. 2 *Order of the operations in a quantum circuit.*

Remark 3.2. A matrix U is unitary if $U^*U = UU^* = I$.

A well-known property of unitary matrices is that they are norm-preserving; that is, given a unitary matrix U and a vector v, $\|Uv\| = \|v\|$. Thus, for a q-qubit system, the quantum state is a unit vector $|\psi\rangle \in \mathbb{C}^{2^q}$, a quantum operation is a matrix $U \in \mathbb{C}^{2^q \times 2^q}$, and the application of U onto the state $|\psi\rangle$ is the unit vector $U|\psi\rangle \in \mathbb{C}^{2^q}$. This leads to the following remarks:

- Quantum operations are *linear*.
- Quantum operations are *reversible*.

While these properties may initially seem to be extremely restrictive, [16] shows that a universal quantum computer is Turing-complete, implying that it can simulate any Turing-computable function with an additional polynomial amount of space, given sufficient time. Out of the two properties indicated above, the most counterintuitive is perhaps reversibility: the classical notion of computation does not appear to be reversible, because memory can be erased and, in the classical Turing machine, symbols can be erased from the tape. However, [7] shows that all computations (including classical computations) can be made reversible by means of extra space. The general idea in making a function invertible is to have separate input and output registers: any output is stored in a different location than the input, so that the input does not have to be erased. This is a standard trick in quantum computing that will be discussed in section 4, but in order to do that, we first need to introduce some notation for quantum circuits.

3.1. Notation for Quantum Circuits. A quantum circuit is represented by indicating which operations are performed on each qubit, or group of qubits. For a quantum computer with q qubits, we represent q qubit lines, where the top line indicates qubit 1 and the rest are given in increasing order from the top. Operations are represented as gates; in what follows, the two terms are used interchangeably. Gates take qubit lines as input, have the same number of qubit lines as output, and apply the unitary matrix indicated on the gate to the quantum state of those qubits. Figure 1 is a simple example. Note that circuit diagrams are read from left to right, but because each gate corresponds to applying a matrix to the quantum state, the matrices corresponding to the gates should be written from right to left in the mathematical expression describing the circuit. For example, in the circuit in Figure 2, the outcome of the circuit is the state $BA|\psi\rangle$, because we start with state $|\psi\rangle$ and first apply the gate with unitary matrix A, followed by B.

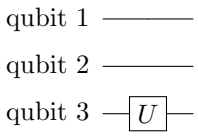


Fig. 3 *A circuit with a single-qubit gate.*

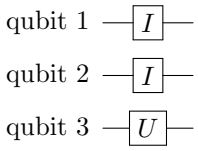


Fig. 4 *Equivalent representation of a circuit with a single-qubit gate.*

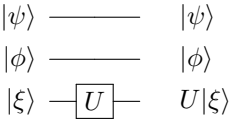


Fig. 5 *Effect of a single-qubit gate on a product state.*

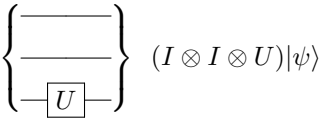


Fig. 6 *Effect of a single-qubit gate on an entangled state.*

Gates can also be applied to individual qubits. Because a single qubit is a vector in $\mathbb{C}^2$, a single-qubit gate is a unitary matrix in $\mathbb{C}^{2\times 2}$. Consider the same 3-qubit device, and suppose we want to apply the gate only to the third qubit. We would write it as in Figure 3. From an algebraic point of view, the action of our first example in Figure 1 on the quantum state is clear: the state of the three qubits is mapped onto another 3-qubit state, as U acts on all the qubits. To understand the example in Figure 3, where U is a single-qubit gate that acts on qubit 3 only, we must imagine that an identity gate is applied to all the empty qubit lines. Therefore, Figure 3 can be thought of as is indicated in Figure 4. This circuit can be interpreted as applying the gate $I \otimes I \otimes U$ to the 3-qubit state. Notice that by convention the matrix U, which is applied to qubit 3, appears in the rightmost term of the tensor product. This is because qubit 3 is associated with the least significant digit according to our little-endian convention; see Definition 1.5 and the subsequent discussion. If we have a product state $|\psi\rangle \otimes |\phi\rangle \otimes |\xi\rangle$, we can write labels as indicated in Figure 5. Indeed, $(I \otimes I \otimes U)(|\psi\rangle \otimes |\phi\rangle \otimes |\xi\rangle) = |\psi\rangle \otimes |\phi\rangle \otimes U|\xi\rangle$. If the system is in an entangled state, however, the action of $(I \otimes I \otimes U)$ cannot be determined in such a simple way, because the state cannot be factored as a product state. Thus, for a general entangled input state, the effect of the circuit is as indicated in Figure 6. Notice that this fact is essentially the reason why simulation of quantum computations on classical computers may take exponential resources in the worst case: to simulate the effect of even a single-qubit gate on the entangled state $|\psi\rangle$, we have to explicitly compute the effect of the $2^q \times 2^q$ matrix $(I \otimes I \otimes U)$ on the state $|\psi\rangle$. This requires exponential space with a naive approach (if the matrices and vectors are stored explicitly), and even with more parsimonious approaches it may require exponential time (e.g., if we

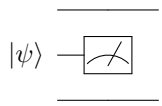


Fig. 7 *Single-qubit measurement.*

compute elements of the state vector one at a time). As long as the quantum state is not entangled, computations can be carried out on each qubit independently, but entanglement requires us to keep track of the full quantum state in 2^q-dimensional complex space, leading to the reguirement for large amounts of memory—or time.

3.2. Input-Output and Measurement Gates. We now discuss the input-output model for quantum computations. The *input* of a quantum algorithm consists of an initial quantum state and a quantum circuit.

Remark 3.3. The quantum state and the quantum circuit must be described in a suitably compact way: for a circuit on q qubits, a unitary matrix can be of size $2^q \times 2^q$, but for an efficient algorithm we require that the circuit contains polynomially many gates in q and that each gate has a compact representation. This will be discussed further in section 3.4.

By convention, the initial quantum state of the quantum computing device is assumed to be $|\vec{0}\rangle_q$ unless otherwise specified. All algorithms described in this paper start from this state. Of course, this does not prevent the algorithm from acting on the state, transforming it into a more suitable one. Examples of how this can be done will be seen in subsequent sections. The important remark is that if there is any data that has to be fed to the algorithm, this data is embedded in the quantum circuit given as part of the input. We should also note that there are hybrid algorithms involving classical and quantum computations. In such situations, the quantum computations can generally be thought of as subroutines, but this does not change the principle that each of these quantum computations will be described by an initial quantum state (typically, $|\vec{0}\rangle_q$) and a quantum circuit. This summarizes the input model. But what is the *output* of the quantum computer?

So far we have characterized properties of quantum states and quantum gates. Remarkably, the state of a q-qubit quantum register is described by a vector of dimension 2^q, exponentially larger than the dimension of the vector required to describe q classical bits. However, there is a catch: in a classical computer we can simply read the state of the bits, whereas in a quantum computer we do not have direct, unrestricted access to the quantum state. Information on the quantum state is only gathered through a measurement gate, indicated in the circuit diagram in Figure 7. We now formally define the effect of a single-bit measurement gate.

ASSUMPTION 3.4. *Information on the state of a quantum computing device can only be obtained through a* measurement. *Given a q-qubit quantum state $|\psi\rangle = \sum_{\vec{\jmath}\in\{0,1\}^q} \alpha_{\vec{\jmath}}|\vec{\jmath}\rangle_q$, a* measurement gate *on qubit k outputs* 0 *with probability $\sum_{\vec{\jmath}\in\{0,1\}^q:\vec{\jmath}_k=0} |\alpha_{\vec{\jmath}}|^2$, and* 1 *with probability $\sum_{\vec{\jmath}\in\{0,1\}^q:\vec{\jmath}_k=1} |\alpha_{\vec{\jmath}}|^2$. Let $x \in \{0,1\}$ be the measured value. After the measurement, the quantum state becomes*

$$\sum_{\substack{\vec{\jmath}\in\{0,1\}^q:\\ \vec{\jmath}_k=x}} \frac{\alpha_{\vec{\jmath}}}{\sqrt{\sum_{\vec{\jmath}:\vec{\jmath}_k=x} |\alpha_{\vec{\jmath}}|^2}} |\vec{\jmath}\rangle_q.$$

The original quantum state is no longer recoverable.

Remark 3.5. The state of the quantum system after a measurement collapses to a linear combination of only those basis states that are consistent with the outcome of the measurement, i.e., basis states $|\vec{\jmath}\rangle$ with $\vec{\jmath}_k = x$. The coefficients $\alpha_{\vec{\jmath}}$ for such basis states are normalized to yield a unit vector.

The rule for single-qubit measurements leads to a very simple and natural expression for the probability of observing a given binary string when measuring all the qubits.

PROPOSITION 3.6. *Given a q-qubit quantum state $|\psi\rangle = \sum_{\vec{\jmath}\in\{0,1\}^q} \alpha_{\vec{\jmath}}|\vec{\jmath}\rangle_q$, applying a measurement gate to the q qubits in any order yields $\vec{\jmath}$ with probability $|\alpha_{\vec{\jmath}}|^2$ for $\vec{\jmath} \in \{0,1\}^q$.*

Proof. We need to show that the probability of observing $\vec{\jmath}$ after q single-qubit measurements is equal to $|\alpha_{\vec{\jmath}}|^2$. We can do this by induction on q. The case $q = 1$ is trivial. We now show how to go from $q-1$ to q. In terms of notation, we will write $\Pr(\mathrm{Q}k \stackrel{M}{=} x)$ to denote the probability that the measurement of qubit k yields $x \in \{0,1\}$. If it is important to indicate the quantum state on which the measurement is performed, we denote it as $\Pr_{|\psi\rangle}(\mathrm{Q}k \stackrel{M}{=} x)$.

Suppose the qubits are measured in an arbitrary order and the qubit in position h is the first to be measured. (The order of the remaining measurements does not matter for the proof, because after the first measurement we will rely on the inductive hypothesis.) The probability of obtaining the outcome $\vec{\jmath}$ is

$$\begin{aligned}
&\Pr_{|\psi\rangle}\left(\mathrm{Q}1 \stackrel{M}{=} \vec{\jmath}_1, \ldots, \mathrm{Q}q \stackrel{M}{=} \vec{\jmath}_q\right) \\
&= \Pr_{|\psi\rangle}\left(\mathrm{Q}1 \stackrel{M}{=} \vec{\jmath}_1, \ldots, \mathrm{Q}(h-1) \stackrel{M}{=} \vec{\jmath}_{h-1}, \mathrm{Q}(h+1) \stackrel{M}{=} \vec{\jmath}_{h+1}, \ldots, \mathrm{Q}q \stackrel{M}{=} \vec{\jmath}_q \,|\, \mathrm{Q}h \stackrel{M}{=} \vec{\jmath}_h\right) \Pr_{|\psi\rangle}\left(\mathrm{Q}h \stackrel{M}{=} \vec{\jmath}_h\right) \\
&= \Pr_{|\phi\rangle}\left(\mathrm{Q}1 \stackrel{M}{=} \vec{\jmath}_1, \ldots, \mathrm{Q}(h-1) \stackrel{M}{=} \vec{\jmath}_{h-1}, \mathrm{Q}(h+1) \stackrel{M}{=} \vec{\jmath}_{h+1}, \ldots, \mathrm{Q}q \stackrel{M}{=} \vec{\jmath}_q\right) \Pr_{|\psi\rangle}\left(\mathrm{Q}h \stackrel{M}{=} \vec{\jmath}_h\right),
\end{aligned}$$

where $|\phi\rangle$ is the state obtained from $|\psi\rangle$ after measuring the qubit in position h and observing $\vec{\jmath}_h$. Therefore, we have

$$|\phi\rangle = \sum_{\substack{\vec{k}\in\{0,1\}^q: \\ \vec{k}_h = \vec{\jmath}_h}} \frac{\alpha_{\vec{k}}}{\sqrt{\sum_{\vec{\ell}\in\{0,1\}^q : \vec{\ell}_h = \vec{\jmath}_h} |\alpha_{\vec{\ell}}|^2}} |\vec{k}\rangle_q := \sum_{\substack{\vec{k}\in\{0,1\}^q: \\ \vec{k}_h = \vec{\jmath}_h}} \beta_{\vec{k}} |\vec{k}\rangle_q,$$

and the coefficients $\beta_{\vec{k}}$, as given above, are only defined for $\vec{k} \in \{0,1\}^q : \vec{k}_h = \vec{\jmath}_h$. By the definition of single-qubit measurement, we also have

$$\Pr_{|\psi\rangle}\left(\mathrm{Q}h \stackrel{M}{=} \vec{\jmath}_h\right) = \sum_{\vec{k}\in\{0,1\}^q : \vec{k}_h = \vec{\jmath}_h} |\alpha_{\vec{k}}|^2.$$

By the induction hypothesis,

$$\Pr_{|\phi\rangle}\left(\mathrm{Q}1 \stackrel{M}{=} \vec{\jmath}_1, \ldots, \mathrm{Q}(h-1) \stackrel{M}{=} \vec{\jmath}_{h-1}, \mathrm{Q}(h+1) \stackrel{M}{=} \vec{\jmath}_{h+1}, \ldots, \mathrm{Q}q \stackrel{M}{=} \vec{\jmath}_q\right) = |\beta_{\vec{\jmath}}|^2,$$

because $|\phi\rangle$ is the state after measuring the hth qubit and obtaining $\vec{\jmath}_h$ as the outcome, and therefore it only contains basis states $\vec{k}$ with $\vec{k}_h = \vec{\jmath}_h$; the induction

Fig. 8 *Multiple-qubit measurement.*

hypothesis means that to compute the probability of observing $\vec{\jmath}_\ell, \ell \neq h$ in the respective positions we simply need to look at the coefficient of the corresponding basis state. Remembering that $\beta_{\vec{k}} = \alpha_{\vec{k}} / \left(\sqrt{\sum_{\vec{\ell}\in\{0,1\}^q:\vec{\ell}_h=\vec{\jmath}_h} |\alpha_{\vec{\ell}}|^2}\right)$, we finally obtain

$$\Pr_{|\psi\rangle}\left(\mathrm{Q1} \overset{M}{=} \vec{\jmath}_1, \ldots, \mathrm{Q}q \overset{M}{=} \vec{\jmath}_q\right) = \frac{|\alpha_{\vec{\jmath}}|^2}{\displaystyle\sum_{\vec{k}\in\{0,1\}^q:\vec{k}_h=\vec{\jmath}_h} |\alpha_{\vec{k}}|^2} \sum_{\substack{\vec{k}\in\{0,1\}^q:\\ \vec{k}_h=\vec{\jmath}_h}} |\alpha_{\vec{k}}|^2 = |\alpha_{\vec{\jmath}}|^2. \qquad \square$$

Proposition 3.6 above shows that the two circuits in Figure 8 are equivalent. In other words, the single-qubit measurement gate is sufficient to apply a measurement on any number of qubits in the most natural way, i.e., the measurement outcome $\vec{\jmath}$ on the q qubits occurs with probability that is exactly equal to the modulus squared of the state coefficient $\alpha_{\vec{\jmath}}$. Notice that with this simple rule, it is easy to compute the probability of obtaining a given string on a given subset of the qubits: we just need to add up the modulus squared of the coefficients for all those basis states that contain the desired string in the desired position.

Example 3.7. Consider again the following 2-qubit state:

$$\alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle = \frac{1}{2}|00\rangle + \frac{1}{2}|01\rangle + \frac{1}{2}|10\rangle + \frac{1}{2}|11\rangle.$$

We previously remarked that this is a product state. As usual, let qubit Q1 be the first qubit (i.e., the one corresponding to the first digit in the two-digit binary strings), and let qubit Q2 be the second qubit (i.e., the one corresponding to the second digit in the two-digit binary strings). Then

$$\Pr(\mathrm{Q1} \overset{M}{=} 0) = |\alpha_{00}|^2 + |\alpha_{01}|^2 = \left(\frac{1}{2}\right)^2 + \left(\frac{1}{2}\right)^2 = \frac{1}{2},$$

$$\Pr(\mathrm{Q1} \overset{M}{=} 1) = |\alpha_{10}|^2 + |\alpha_{11}|^2 = \left(\frac{1}{2}\right)^2 + \left(\frac{1}{2}\right)^2 = \frac{1}{2},$$

$$\Pr(\mathrm{Q2} \overset{M}{=} 0) = |\alpha_{00}|^2 + |\alpha_{10}|^2 = \left(\frac{1}{2}\right)^2 + \left(\frac{1}{2}\right)^2 = \frac{1}{2},$$

$$\Pr(\mathrm{Q2} \overset{M}{=} 1) = |\alpha_{01}|^2 + |\alpha_{11}|^2 = \left(\frac{1}{2}\right)^2 + \left(\frac{1}{2}\right)^2 = \frac{1}{2}.$$

Suppose we measure Q2 and obtain 1 as the outcome of the measurement. Then the state of the 2-qubit system collapses to

$$\frac{1}{\sqrt{2}}|01\rangle + \frac{1}{\sqrt{2}}|11\rangle.$$

The outcome distribution for Q1 for this new state is

$$\Pr(\text{Q1} \overset{M}{=} 0) = \frac{1}{2}, \qquad \Pr(\text{Q1} \overset{M}{=} 1) = \frac{1}{2}.$$

Hence, the probability of observing 0 or 1 when measuring qubit Q1 did not change after the measurement.

Consider now the following entangled 2-qubit state:

$$\beta_{00}|00\rangle + \beta_{11}|11\rangle = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle.$$

Doing the calculations, we still have

$$\Pr(\text{Q1} \overset{M}{=} 0) = |\beta_{00}|^2 = \frac{1}{2}, \qquad \Pr(\text{Q1} \overset{M}{=} 1) = |\beta_{11}|^2 = \frac{1}{2},$$

$$\Pr(\text{Q2} \overset{M}{=} 0) = |\beta_{00}|^2 = \frac{1}{2}, \qquad \Pr(\text{Q2} \overset{M}{=} 1) = |\beta_{11}|^2 = \frac{1}{2}.$$

Suppose we measure qubit Q2 and obtain 1 as the outcome of the measurement. Then the state of the 2-qubit system collapses to

$$|11\rangle.$$

If we measure Q1 from this state, obtain

$$\Pr(\text{Q1} \overset{M}{=} 0) = 0, \qquad \Pr(\text{Q1} \overset{M}{=} 1) = 1.$$

The situation is now very different: the probabilities of the outcomes from a measurement on Q1 have changed after measuring Q2. This is exactly the concept of entanglement: when two or more qubits are entangled, they affect each other, and applying a measurement on one qubit changes the probability distribution for the other qubits.

The example above can be seen in terms of conditional probabilities: if, for all $x, y \in \{0, 1\}$, we have $\Pr(\text{Q1} \overset{M}{=} x) = \Pr(\text{Q1} \overset{M}{=} x | \text{Q2} \overset{M}{=} y)$, then the two qubits are not entangled (product state), whereas if $\Pr(\text{Q1} \overset{M}{=} x) \neq \Pr(\text{Q2} \overset{M}{=} x | \text{Q2} \overset{M}{=} y)$ for some x, y, there is entanglement. Indeed, recall from Example 1.3 that taking the tensor product of two vectors containing outcome probabilities for independent random variables yields the joint probability distribution. Quantum state vectors do not contain outcome probabilities, but the modulus squared of the components of the state vector corresponds to a probability. Furthermore, for any two complex numbers $\alpha, \beta \in \mathbb{C}$ we have $|\alpha\beta|^2 = |\alpha|^2|\beta|^2$, so the operation applied to compute probabilities from state coefficients is distributive with respect to multiplication. A product state is a tensor product of smaller-dimensional state vectors, and hence it leads to outcome probabilities that are simply the product of the outcome probabilities corresponding to measuring each of the qubits independently. Conversely, an entangled state is not

a product state, and the outcomes of measuring each of the qubits are no longer independent.

Remark 3.8. Despite the above discussion, it would be wrong to think of the quantum state as a probability distribution: the quantum state *induces* a probability distribution by taking the modulus squared of its entries, but it is not a probability distribution! Indeed, the coefficients in a quantum state are complex numbers unrestricted in sign, while probabilities are nonnegative real numbers. Furthermore, just as there is an infinite set of complex numbers that have the same modulus (i.e., the set $\{a \in \mathbb{C} : |a| = v\}$ for some real number $v > 0$ is infinite), there is an infinite number of quantum state vectors in $(\mathbb{C}^2)^{\otimes q}$ that yield the same distribution. Some of these may yield the same outcome of the computation, but others may not: this is discussed in the next two examples.

Example 3.9. Suppose we have two q-qubit quantum states $|\psi\rangle, |\phi\rangle$ satisfying $|\psi\rangle = e^{i\theta}|\phi\rangle$ for some $\theta \in \mathbb{R}$. Now consider the application of some unitary matrix U onto $|\psi\rangle$ and $|\phi\rangle$, followed by a measurement of all the qubits. Define

$$U|\phi\rangle := \sum_{\vec{j}\in\{0,1\}^q} \alpha_{\vec{j}}|\vec{j}\rangle$$

for some (normalized) coefficients $\alpha_{\vec{j}}$, which implies

$$U|\psi\rangle = Ue^{i\theta}|\phi\rangle = \sum_{\vec{j}\in\{0,1\}^q} e^{i\theta}\alpha_{\vec{j}}|\vec{j}\rangle.$$

This means that for a given $\vec{k}$,

$$\Pr_{|\phi\rangle}(\mathrm{Q1} \overset{M}{=} \vec{k}_1, \ldots, \mathrm{Q}q \overset{M}{=} \vec{k}_q) = |\alpha_{\vec{k}}|^2, \qquad \Pr_{|\psi\rangle}(\mathrm{Q1} \overset{M}{=} \vec{k}_1, \ldots, \mathrm{Q}q \overset{M}{=} \vec{k}_q) = |e^{i\theta}\alpha_{\vec{k}}|^2 = |\alpha_{\vec{k}}|^2,$$

so the probability of obtaining $\vec{k}$ as the outcome of a measurement is the same for both $|\psi\rangle$ and $|\phi\rangle$. Since this is true after applying an arbitrary unitary U, it is also true after applying a whole circuit, which is just a sequence of unitaries. Hence, if the vectors $|\psi\rangle, |\phi\rangle$ satisfy the relationship $|\psi\rangle = e^{i\theta}|\phi\rangle$, they induce the same outcome distribution. The factor $e^{i\theta}$ is usually called the *global phase* and can be safely be ignored.

Example 3.10. Consider the following two 1-qubit state vectors:

$$|\psi\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle, \qquad |\phi\rangle = \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle.$$

Both induce the same probability distribution on the measurement outcomes:

$$\Pr_{|\psi\rangle}(\mathrm{Q1} \overset{M}{=} 0) = \frac{1}{2}, \qquad \Pr_{|\psi\rangle}(\mathrm{Q1} \overset{M}{=} 1) = \frac{1}{2},$$
$$\Pr_{|\phi\rangle}(\mathrm{Q1} \overset{M}{=} 0) = \frac{1}{2}, \qquad \Pr_{|\phi\rangle}(\mathrm{Q1} \overset{M}{=} 1) = \frac{1}{2}.$$

But $|\psi\rangle$ and $|\phi\rangle$ are very different states! If we apply a certain unitary matrix to both (this matrix is called the Hadamard gate, as we will see in section 3.4), we obtain

very different results—orthogonal vectors, in fact:

$$\frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}|\psi\rangle = \frac{1}{2}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}\begin{pmatrix}1\\ 1\end{pmatrix} = \frac{1}{2}\begin{pmatrix}2\\ 0\end{pmatrix} = |0\rangle,$$
$$\frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}|\phi\rangle = \frac{1}{2}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}\begin{pmatrix}1\\ -1\end{pmatrix} = \frac{1}{2}\begin{pmatrix}0\\ 2\end{pmatrix} = |1\rangle.$$

This illustrates the danger of thinking about the quantum state as a probability distribution.

3.3. The No-Cloning Principle. Because measurement destroys the quantum state, it is natural to look for a way to create a copy of a quantum state. If a clone could be created, it would be possible to perform measurements on the clone, so that the original state would not be destroyed. Furthermore, cloning would allow us to take several measurements of the same set of qubits without having to repeat the circuit that creates the quantum state. However, it turns out that cloning is impossible: this is a direct consequence of the properties of quantum gates, in particular the fact that gates are unitary matrices.

PROPOSITION 3.11. *Let $|\psi\rangle$ be an arbitrary quantum state on q qubits. There does not exist a unitary matrix that maps $|\psi\rangle_q \otimes |\vec{0}\rangle_q$ to $|\psi\rangle_q \otimes |\psi\rangle_q$.*

Proof. Suppose there exists such a unitary U. Then for any two quantum states $|\psi\rangle_q, |\phi\rangle_q$, we have

$$U(|\psi\rangle_q \otimes |\vec{0}\rangle_q) = |\psi\rangle_q \otimes |\psi\rangle_q,$$
$$U(|\phi\rangle_q \otimes |\vec{0}\rangle_q) = |\phi\rangle_q \otimes |\phi\rangle_q.$$

Using these equalities, and remembering that $U^*U = I$, we can write

$$\langle\phi|\psi\rangle = \langle\phi|\psi\rangle\langle\vec{0}|\vec{0}\rangle = \langle\phi|\psi\rangle \otimes \left(\langle\vec{0}|\vec{0}\rangle\right) = (\langle\phi|_q \otimes \langle\vec{0}|_q)(|\psi\rangle_q \otimes |\vec{0}\rangle_q),$$
$$= (\langle\phi|_q \otimes \langle\vec{0}|_q)U^*U(|\psi\rangle_q \otimes |\vec{0}\rangle_q) = (\langle\phi|_q \otimes \langle\phi|_q)(|\psi\rangle_q \otimes |\psi\rangle_q) = \langle\phi|\psi\rangle^2.$$

But $\langle\phi|\psi\rangle = \langle\phi|\psi\rangle^2$ is only true if $\langle\phi|\psi\rangle$ is equal to 0 or to 1, contradicting the fact that $|\phi\rangle, |\psi\rangle$ are arbitrary quantum states. ∎

The above proposition shows that we cannot copy an arbitrary quantum state. We remark that the proof does not rule out the possibility of constructing an operation that copies a specific quantum state. In other words, if we knew what quantum state we wanted to copy, we could construct a unitary matrix to do that; but it is impossible to construct a single unitary matrix to copy all possible states. This establishes that we cannot "cheat" the destructive effect of a measurement by simply cloning the state before the measurement. Hence, whenever we run a circuit that produces an output quantum state, in general we can reproduce the output quantum state only by repeating all the steps of the algorithm.

3.4. Basic Operations and Universality. Quantum computation does not allow the user to specify just any unitary matrix in the code, just as classical computations do not allow the user to specify any classical function. Rather, the user is limited to gates (unitary matrices) that are efficiently specifiable and implementable, just as classically one can only write efficient programs by specifying a polynomial-size

sequence of basic operations on bits. The specification of a unitary matrix must be done by combining gates out of a basic set, which can be thought of as the instruction set of the quantum computer. We will now discuss what these basic gates are and how they can be combined to form other operations.

The first operations that we discuss are the *Pauli gates*.

DEFINITION 3.12. *The four Pauli gates are the following single-qubit gates:*

$$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \qquad X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \qquad Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \qquad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}.$$

PROPOSITION 3.13. *The Pauli gates form a basis for $\mathbb{C}^{2\times 2}$, they are Hermitian, and they satisfy the relationship $XYZ = iI$.*

The proof is left as an exercise. The X gate is the equivalent of a NOT gate in classical computers, as it implements a bit (rather, qubit) flip, changing from $|0\rangle$ to $|1\rangle$ and vice versa:

$$X|0\rangle = |1\rangle, \qquad X|1\rangle = |0\rangle.$$

The Z gate is also called a phase flip gate: it leaves $|0\rangle$ unchanged and maps $|1\rangle$ to $-|1\rangle$:

$$Z|0\rangle = |0\rangle, \qquad Z|1\rangle = -|1\rangle.$$

A single-qubit gate that is used in many quantum algorithms is the so-called Hadamard gate:

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

The action of H is as follows:

$$H|0\rangle = \frac{1}{\sqrt{2}} \left(|0\rangle + |1\rangle \right), \qquad H|1\rangle = \frac{1}{\sqrt{2}} \left(|0\rangle - |1\rangle \right).$$

In subsequent sections we will need an algebraic expression for the action of Hadamard gates on basis states. The effect of H on a 1-qubit basis state $|x\rangle$ (where $x = 0$ or 1) can be summarized as follows:

$$H|x\rangle = \frac{1}{\sqrt{2}}(|0\rangle + (-1)^x|1\rangle) = \frac{1}{\sqrt{2}} \sum_{k=0}^{1} (-1)^{kx}|k\rangle.$$

This is consistent with our previous definition. If we apply $H^{\otimes q}$ on a q-qubit basis state $|\vec{x}\rangle_q$, we obtain

$$\begin{aligned} H^{\otimes q}|\vec{x}\rangle_q &= \frac{1}{\sqrt{2^q}} \sum_{k_1=0}^{1} \cdots \sum_{k_q=0}^{1} (-1)^{\sum_{h=1}^{q} k_h \vec{x}_h} |k_1\rangle_1 \otimes \cdots \otimes |k_q\rangle_1 \\ &= \frac{1}{\sqrt{2^q}} \sum_{\vec{k} \in \{0,1\}^q} (-1)^{\vec{k} \bullet \vec{x}} |\vec{k}\rangle_q, \end{aligned} \tag{3.1}$$

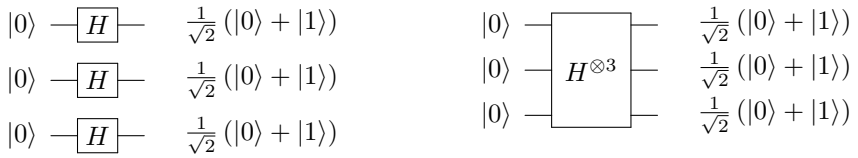


Fig. 9 *Two representations for multiple Hadamard gates.*

where $\bullet$ is the bitwise dot product, as defined in section 1.3. When considering multiple Hadamard gates in parallel, we will also make use of the following relationship, which can be easily verified using the definition:

$$H^{\otimes q} = \frac{1}{\sqrt{2}} \begin{pmatrix} H^{\otimes q-1} & H^{\otimes q-1} \\ H^{\otimes q-1} & -H^{\otimes q-1} \end{pmatrix}. \tag{3.2}$$

The next proposition shows one of the reasons why the Hadamard gate is frequently employed in many quantum algorithms.

PROPOSITION 3.14. *Given a q-qubit quantum computing device initially in the state $|\vec{0}\rangle_q$, applying the Hadamard gate to all qubits, or equivalently the matrix $H^{\otimes q}$, yields the uniform superposition of basis states $\frac{1}{\sqrt{2^q}} \sum_{\vec{\jmath} \in \{0,1\}^q} |\vec{\jmath}\rangle$.*

Proof. We have:

$$H^{\otimes q}|\vec{0}\rangle_q = H^{\otimes q}|0\rangle^{\otimes q} = (H|0\rangle)^{\otimes q} = \left(\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle\right)^{\otimes q} = \frac{1}{\sqrt{2^q}} \sum_{\vec{\jmath} \in \{0,1\}^q} |\vec{\jmath}\rangle. \qquad \square$$

Remark 3.15. The uniform superposition of the 2^q basis states on q qubits can be obtained from the initial state $|\vec{0}\rangle_q$ by applying q gates only.

The multiple Hadamard can be represented by one of the equivalent circuits given in Figure 9. Many quantum algorithms (for example, the algorithms discussed in sections 4 and 5) start by setting the state of the quantum device to a uniform superposition, and then apply further operations which, by linearity, are simultaneously applied to all the possible binary strings. This is a remarkable advantage of quantum computing over classical computing.

Readers with advanced knowledge of theoretical computer science might be wondering how this compares to classical probabilistic computation: after all, probabilistic Turing machines are a well-known concept in computational complexity. A probabilistic Turing machine is initialized with a set of random bits that take an unknown value and influence the state transition. The state is described by a probability distribution over all the possible states, because we do not know the value of the random bits with which the machine is initialized. When a state transition occurs, to update the description of the state we need to apply the transition to all states that appear with positive probability. In this sense, operations in a probabilistic Turing machine can be thought of as being simultaneously applied to many (possibly all) binary strings. However, a probabilistic Turing machine admits a more compact description of the state: if we know the random bits with which the machine is initialized, then the state becomes deterministically known. Hence, for a given value of the random bits, the state of the probabilistic Turing machine

Fig. 10 *The $CNOT_{12}$, or controlled NOT, gate with control qubit* 1 *and target qubit* 2.

can be described in linear space, and operations map one state into another state. On the other hand, it is not known how to obtain such a compact description for a quantum computer: there is no equivalent for the random bits, and a characterization of the state truly requires an exponential number of complex coefficients. In fact, it is believed that quantum computers are more powerful than probabilistic Turing machines, although there is no formal proof.

To conclude our discussion on single-qubit gates, we remark that all of them can be represented by the following parameterized matrix that describes all unitary matrices (up to a global phase factor; see Example 3.9):

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

All single-qubit gates can be obtained by an appropriate choice of parameters θ, ϕ, λ. The open-source library Qiskit used for the numerical experiments in section 6 allows for specifying a single-qubit gate in terms of θ, ϕ, λ, if so desired.

Another fundamental gate is the CNOT gate, called "controlled NOT." The CNOT gate is a 2-qubit gate that has a control bit and a target bit and acts as follows: if the control bit is $|0\rangle$, nothing happens, whereas if the control bit is $|1\rangle$, the target bit is bit-flipped (i.e., the same effect as the X gate). The corresponding circuit is given in Figure 10. The matrix description of the gate with control qubit 1 and target qubit 2 is as follows:

$$\mathrm{CNOT}_{12} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}.$$

We can easily see that the effect of CNOT is as follows:

$$\mathrm{CNOT}_{12}|00\rangle = |00\rangle, \qquad \mathrm{CNOT}_{12}|01\rangle = |01\rangle,$$
$$\mathrm{CNOT}_{12}|10\rangle = |11\rangle, \qquad \mathrm{CNOT}_{12}|11\rangle = |10\rangle.$$

An interesting feature of the CNOT gate is that it can be used to swap two qubits. A swap between two qubits Qi and Qj is defined as the operation that maps a quantum state into a new quantum state in which every basis state has its ith and jth digits permuted. If two qubits are in a product state $|\psi\rangle_1 \otimes |\phi\rangle_1$, then $\mathrm{SWAP}(|\psi\rangle_1 \otimes |\phi\rangle_1) = |\phi\rangle_1 \otimes |\psi\rangle_1$. Considering that CNOT, like all quantum gates, is a linear map, it may sound surprising that it can implement a swap. However, the SWAP gate can indeed be constructed out of CNOTs, as depicted in Figure 11.

PROPOSITION 3.16. *The circuit in Figure* 11*, constructed with three CNOTs, swaps qubits* 1 *and* 2*.*

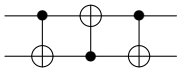


Fig. 11 *A circuit that swaps two qubits.*

Proof. By linearity, it suffices to show that the circuit above maps $|00\rangle \to |00\rangle$, $|01\rangle \to |10\rangle$, $|10\rangle \to |01\rangle$, and $|11\rangle \to |11\rangle$. We have

$$\begin{aligned}
\mathrm{CNOT}_{12}\mathrm{CNOT}_{21}\mathrm{CNOT}_{12}|00\rangle &= \mathrm{CNOT}_{12}\mathrm{CNOT}_{21}|00\rangle = \mathrm{CNOT}_{12}|00\rangle = |00\rangle,\\
\mathrm{CNOT}_{12}\mathrm{CNOT}_{21}\mathrm{CNOT}_{12}|01\rangle &= \mathrm{CNOT}_{2}1\mathrm{CNOT}_{21}|01\rangle = \mathrm{CNOT}_{12}|11\rangle = |10\rangle,\\
\mathrm{CNOT}_{12}\mathrm{CNOT}_{21}\mathrm{CNOT}_{12}|10\rangle &= \mathrm{CNOT}_{12}\mathrm{CNOT}_{21}|11\rangle = \mathrm{CNOT}_{12}|01\rangle = |01\rangle,\\
\mathrm{CNOT}_{12}\mathrm{CNOT}_{21}\mathrm{CNOT}_{12}|11\rangle &= \mathrm{CNOT}_{12}\mathrm{CNOT}_{21}|10\rangle = \mathrm{CNOT}_{12}|10\rangle = |11\rangle.
\end{aligned}$$

Therefore, the SWAP circuit maps

$$\alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle \to \alpha_{00}|00\rangle + \alpha_{01}|10\rangle + \alpha_{10}|01\rangle + \alpha_{11}|11\rangle. \quad \square$$

The SWAP circuit is particularly important for practical reasons: in the current generation of quantum computing hardware, 2-qubit gates can only be applied among certain pairs of qubits. For example, when employing one of the most prevalent quantum hardware technologies (superconducting qubits; see, e.g., [12, 18]), 2-qubit gates can only be applied to qubits that are physically adjacent on a chip. Thanks to the SWAP, as long as the connectivity graph of the qubits on the device is a connected graph, 2-qubit gates can be applied to any pair of qubits: if the qubits are not directly connected on the graph (e.g., physically located next to each other on the chip), we just need to SWAP one of them as many times as is necessary to bring it to a location adjacent to the other qubit. This way, we can assume that each qubit can interact with all other qubits from a theoretical point of view, even if from a practical perspective this may require extra SWAP gates.

A set of gates consisting of (some) single-qubit gates plus CNOT can be shown to be sufficient to construct any unitary matrix with arbitrary precision. This is the concept of *universality*.

DEFINITION 3.17. *A unitary matrix A is an ϵ-approximation of a unitary matrix U if $\sup_{\psi:\|\psi\|=1} \|(U-A)\psi\| < \epsilon$. A set of gates that can be used to construct an ϵ-approximation of any unitary matrix, for any $\epsilon > 0$ and on any given number of qubits, is called a* universal *set of gates.*

To build a universal set of gates, the first step is to show how to construct arbitrary single-qubit gates, and the next is to use these gates to build larger ones.

THEOREM 3.18 (Solovay–Kitaev [25, 30]). *Let $U \in \mathbb{C}^{2\times 2}$ be an arbitrary unitary matrix. Then there exists a constant c such that there exists a sequence S of gates of length $O(\log^c \frac{1}{\epsilon})$ that yields an ϵ-approximation of U and consists only of H, $T = \left(\begin{smallmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{smallmatrix}\right)$ and CNOT gates.*

The theorem implies that just two single-qubit gates together with CNOT allow us to build any single-qubit gate with arbitrary precision. [15] gives a proof with $c \approx 3.98$. More recent work gives an improved algorithm with smaller c, in fact, even $c = 1$ (but different constants); see [34, 26]. To go from single-qubit gates to general q-qubit gates, one needs at most $O(q^2 4^q)$ basic gates (i.e., the gates of Theorem 3.18);

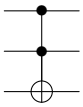


Fig. 12 *The CCNOT, or doubly-controlled NOT, gate.*

intuitively, this is because each gate on q qubits has $2^q \times 2^q$ elements, and it takes q^2 basic gates to "fill" an arbitrary element of a large matrix—for a detailed discussion, see [30, Chap. 4]. In other words, the set of gates consisting of just H, T, and CNOT is universal. This shows that with a very small set of basic gates, we can construct any unitary matrix in any dimension, although this may require many operations. Once again, this is important for practical reasons: the current generation of quantum hardware only allows (natively) certain single-qubit gates and CNOT gates, but all other gates can be constructed from these ones.

We conclude our discussion on basic operations with a quantum circuit for the logic AND gate. Although this gate is not used by the algorithms described later, it is used in the numerical examples. We already know that the X gate performs the logic NOT: having access to the AND guarantees that we can construct any Boolean circuit (since we have already stated in section 3 that universal quantum computers are Turing-complete, they can of course perform Boolean logic). The quantum version of the AND gate is the CCNOT (doubly-controlled NOT) gate, which acts on three qubits: it has two control qubits and it flips (bit flip, i.e., as the X gate) the third qubit if and only if both control qubits are $|1\rangle$. The gate is depicted in Figure 12. The action of CCNOT can be described as $|x\rangle_1 \otimes |y\rangle_1 \otimes |z\rangle_1 \to |x\rangle_1 \otimes |y\rangle_1 \otimes |z \oplus (x \cdot y)\rangle_1$, where $x, y, z \in \{0, 1\}$. Notice that if $z = 0$, CCNOT indeed computes the logical AND between x and y because $0 \oplus (x \cdot y) = x \wedge y$.

Of course, following our earlier discussion, CCNOT can be constructed using only the basic gates indicated in Theorem 3.18. For this, we can use the circuit in Figure 13; see [30]. In this circuit we also use the conjugate transpose T^* of the T gate, but it is easy to see that if we really want to stick to the gates H, T, and CNOT only, T^* can be constructed from T because $e^{-i\frac{\pi}{4}} = e^{i\frac{7\pi}{8}}$. Verifying correctness of the construction in Figure 13 requires a few calculations that we leave as an exercise. One way is to carry out the matrix multiplications; another way, probably more manageable if doing calculations by hand, is to use linearity and look at the effect of the circuit on each of the 2^3 possible basis states. We show only part of the calculations here. Suppose the circuit is applied to the basis state $|11x\rangle$ with $x \in \{0, 1\}$. After performing several simplifications (T and T^* cancel out, the T gate has no effect on a qubit in state $|0\rangle$, and we can transform the CNOTs on the third qubit line into X gates because we already know that the first and second qubits are in state $|1\rangle$), we find out that the circuit maps

$$|1\rangle \otimes |1\rangle \otimes |x\rangle \to (T|1\rangle) \otimes (T|1\rangle) \otimes (HTXT^*XTXT^*XH|x\rangle).$$

Doing the calculations, we see that

$$HTXT^*XTXT^*XH = \begin{pmatrix} 0 & -i \\ -i & 0 \end{pmatrix},$$

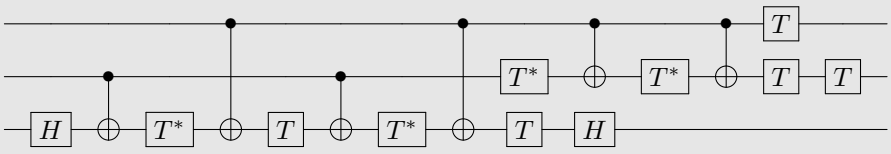


Fig. 13 *Decomposition of CCNOT in terms of the universal set of gates of Theorem* 3.18.

so that the mapping reads

$$\begin{aligned}|1\rangle \otimes |1\rangle \otimes |1\rangle \rightarrow &(T|1\rangle) \otimes (T|1\rangle) \otimes (HTXT^*XTXT^*XH|1\rangle) \\ &= (e^{i\frac{\pi}{4}}|1\rangle) \otimes (e^{i\frac{\pi}{4}}|1\rangle) \otimes (-i|0\rangle) = |1\rangle \otimes |1\rangle \otimes |0\rangle, \\ |1\rangle \otimes |1\rangle \otimes |0\rangle \rightarrow &(e^{i\frac{\pi}{4}}|1\rangle) \otimes (e^{i\frac{\pi}{4}}|1\rangle) \otimes (-i|1\rangle) = |1\rangle \otimes |1\rangle \otimes |1\rangle.\end{aligned}$$

In general, coming up with these constructions requires a good deal of experience, or a piece of code implementing the algorithms referenced above.

3.5. Can We Solve NP-Hard Problems? It is important to remark that even if we can easily create a uniform superposition of all basis states, the rules of measurement imply that using just this easily-obtained superposition does not allow us to satisfactorily solve NP-complete problems such as, for example, SAT (the satisfiability problem). Indeed, suppose we have a quantum circuit U_f that encodes a SAT formula on q Boolean variables; in other words, a unitary $U_f : |\vec{\jmath}\rangle_q \otimes |0\rangle_1 \rightarrow |\vec{\jmath}\rangle_q \otimes |f(\vec{\jmath})\rangle_1$, where $f(\vec{\jmath})$ is 1 if the binary string $\vec{\jmath}$ satisfies the formula, and 0 if not. We might be tempted to apply $H^{\otimes q}$ to the initial state $|\vec{0}\rangle_q$ to create the uniform superposition $\frac{1}{\sqrt{2^q}} \sum_{\vec{\jmath} \in \{0,1\}^q} |\vec{\jmath}\rangle$, apply U_f to this superposition (which evaluates the truth assignment of all possible binary strings), and then perform a measurement on all $q+1$ qubits. But measuring the state,

$$U_f \left(\frac{1}{\sqrt{2^q}} \sum_{\vec{\jmath} \in \{0,1\}^q} |\vec{\jmath}\rangle \otimes |0\rangle_1 \right) = \frac{1}{\sqrt{2^q}} \sum_{\vec{\jmath} \in \{0,1\}^q} |\vec{\jmath}\rangle \otimes |f(\vec{\jmath})\rangle,$$

will return a binary string that satisfies the formula if and only if the last qubit has value 1 after the measurement, and this happens with a probability that depends on the number of binary assignments that satisfy the formula. If the SAT problem at hand is solved by exactly ρ assignments out of 2^n possible assignments, then the probability of finding the solution after one measurement is $\frac{\rho}{2^n}$: we have done nothing more than randomly sampling a binary string and hoping that it satisfies the SAT formula. Clearly, this is not a good algorithm. In fact, in general solving NP-hard problems in polynomial time with quantum computers is not believed to be possible: most researchers believe that the complexity class BQP, which is the class of problems solvable in polynomial time by a quantum computer with bounded (and small) error probability, does not contain the class NP. Of course, one cannot hope to prove this unconditionally, because showing NP $\not\subseteq$ BQP would resolve the famous P vs. NP problem. Nevertheless, it is strongly believed that NP $\not\subseteq$ BQP, due to the lower bound on black-box search of [8] and the inability of quantum computing researchers to develop an efficient quantum algorithm for SAT.

Even if we cannot solve all difficult problems in polynomial time using a quantum computer, we will see in the next sections two examples of quantum algorithms that are faster than any known classical algorithm. The basic principle employed by these algorithms is to start with a uniform superposition of basis states, then apply operations that make the basis states interact with each other so that the modulus of the coefficients for some (desirable) basis states increase, which implies that the other coefficients decrease. Performing a measurement will then reveal the solution to the problem at hand, or some useful information about the solution, with high probability. Of course, how to do this in order to solve a specific problem is exactly where the crux of the matter lies.

4. A Simple Period Finding Problem: Simon's Algorithm. In this section we describe a quantum algorithm, known as Simon's algorithm [36], that gives an expected exponential speedup with respect to classical algorithms. Simon's algorithm is one of the first examples of quantum speedup presented in the literature; other early examples are [17, 9]. However, we discuss Simon's algorithm because it has many interesting features from an educational perspective (namely, the fact that it uses both classical and quantum computation, and that it yields an exponential speedup).

Admittedly, the problem that Simon's algorithm solves is not very useful, but the ideas shown here give us a flavor of what quantum computation can do. In fact, this algorithm was an inspiration for the well-known and groundbreaking work of Shor on integer factorization [35]: a large part of Shor's algorithm relies on the solution of a period finding problem, and Simon's algorithm solves a simplified problem of the same flavor. Shor's algorithm is, however, much more involved than Simon's algorithm, and a full treatment requires several number-theoretical results that are beyond the scope of this introductory material. Thus, we will focus on Simon's algorithm, but by the end of this paper readers should be capable of undertaking the study of Shor's algorithm on their own, see also section 7.

For Simon's algorithm, we are told that there exists a function $f : \{0,1\}^n \to \{0,1\}^n$ with the property that $f(\vec{x}) = f(\vec{z})$ if and only if $\vec{x} = \vec{z} \oplus \vec{a}$ for some unknown (but fixed) $\vec{a} \in \{0,1\}^n$. We do not know anything else about the function, and the goal is to find $\vec{a}$ by querying the function the smallest number of times.

Remark 4.1. For both Simon's algorithm and Grover's algorithm, the complexity of an algorithm is determined only in terms of the number of calls to the function f. Considerations on what the function f actually implements, and the number of operations that are performed *inside* of f, or between the calls to f, are not part of our analysis. This model is known as *query complexity*, because—as the name implies—it defines the complexity of an algorithm as the number of queries to a given function (in this case, f). Query complexity is used as a model to answer important theoretical questions. There are many quantum algorithms that yield speedups under the query complexity model, but some others, e.g., Shor's algorithm, are faster than (known) classical algorithms under the more traditional computational complexity model, i.e., number of basic operations.

Notice that if $\vec{a} = \vec{0}$, then the function is one-to-one, whereas if $\vec{a} \neq \vec{0}$ the function is two-to-one, because for every $\vec{x}$ there is exactly one other number in the domain for which the function has the same value. The function f is assumed to be given as a quantum circuit on $q = 2n$ qubits, via the unitary U_f depicted in Figure 14, and we are allowed to query the function in superposition. Remember that by linearity, to describe the effect of U_f it is sufficient to describe its behavior on all basis states.

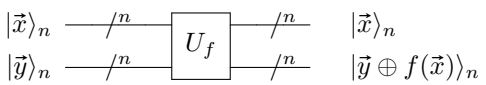


Fig. 14 *The circuit implementing U_f for Simon's problem, with basis states $\vec{x}, \vec{y} \in \{0,1\}^n$.*

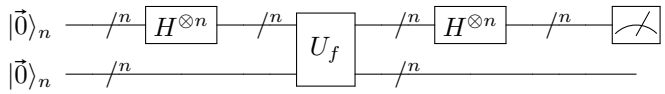


Fig. 15 *Quantum circuit used in Simon's algorithm.*

This particular form of the function, which maps $|\vec{x}\rangle_n \otimes |\vec{y}\rangle_n$ to $|\vec{x}\rangle_n \otimes |\vec{y} \oplus f(\vec{x})\rangle_n$, is typical of the quantum world. Notice that if $\vec{y} = \vec{0}$, then $|\vec{y} \oplus f(\vec{x})\rangle_n = |f(\vec{x})\rangle_n$ so the circuit computes the desired function. Furthermore, this is a reversible function, because applying the same circuit U_f goes back to the initial state:

$$U_f U_f(|\vec{x}\rangle_n \otimes |\vec{y}\rangle_n) = U_f(|x\rangle_n \otimes |\vec{y} \oplus f(\vec{x})\rangle_n) = |\vec{x}\rangle_n \otimes |\vec{y} \oplus f(\vec{x}) \oplus f(\vec{x})\rangle_n = |\vec{x}\rangle_n \otimes |\vec{y}\rangle_n.$$

4.1. Classical Algorithm. Because we do not know anything about the binary string $\vec{a}$, the best we can do is to feed inputs to the function and try to extract information from the output. The number $\vec{a}$ is determined once we find two distinct inputs $\vec{x}, \vec{z}$ such that $f(\vec{x}) = f(\vec{z})$, because then $\vec{x} = \vec{z} \oplus \vec{a}$, which implies $\vec{x} \oplus \vec{z} = \vec{a}$.

Suppose we have evaluated m distinct input values and we have not found a match. Then $\vec{a} \neq \vec{x} \oplus \vec{z}$ for all $\vec{x}, \vec{z}$ previously evaluated, and therefore we have eliminated at most $m(m-1)/2$ values of $\vec{a}$. (Fewer values might be eliminated if we test inputs equal to $\vec{x} \oplus \vec{y} \oplus \vec{z}$ for any three input values $\vec{x}, \vec{y}, \vec{z}$ already tested. In fact, if we test $\vec{w}$ such that $\vec{w} = \vec{x} \oplus \vec{y} \oplus \vec{z}$, we have that $\vec{w} \oplus \vec{z} = \vec{x} \oplus \vec{y}$, and therefore the value $\vec{w} \oplus \vec{z}$ has already been eliminated from the list of possible values of $\vec{a}$.) Since $m(m-1)/2$ is small compared to 2^n, the probability of success $\frac{m(m-1)}{2^{n+1}}$ is very small until we have evaluated a number of inputs that is in the order of 2^n. In particular, to guarantee a probability of success of at least ρ, we need $m(m-1) \geq \rho 2^{n+1}$, which implies that $m \in O(\sqrt{\rho 2^n})$. Hence, for any positive constant ρ, the number of required iterations is exponential. After evaluating $\frac{1+\sqrt{2^{n+3}+1}}{2} \in O(2^{n/2})$ distinct input values satisfying the condition outlined above for nonmatching triplets (to obtain this number, we found the smallest value of m such that $m(m-1) \geq 2^{n+1}$), we are guaranteed that a matching pair has been found, or we can safely determine that $\vec{a} = \vec{0}$.

4.2. Simon's Algorithm: Quantum Computation. Using a quantum computer, we can determine $\vec{a}$ much faster. The idea, first described in [36], is to apply the circuit in Figure 15. From an algebraic point of view, the circuit is described by the following equation:

$$(H^{\otimes n} \otimes I^{\otimes n}) U_f (H^{\otimes n} \otimes I^{\otimes n})(|\vec{0}\rangle_n \otimes |\vec{0}\rangle_n).$$

We now analyze the output of the quantum circuit by looking at the quantum states at intermediate steps of the circuit. Let $|\psi\rangle$ be the state just before the U_f gate, $|\phi\rangle$ the state just after U_f, and $|\chi\rangle$ the final state. In other words,

$$\begin{aligned}
|\psi\rangle &= (H^{\otimes n} \otimes I^{\otimes n})(|\vec{0}\rangle_n \otimes |\vec{0}\rangle_n), \\
|\phi\rangle &= U_f(H^{\otimes n} \otimes I^{\otimes n})(|\vec{0}\rangle_n \otimes |\vec{0}\rangle_n), \\
|\chi\rangle &= (H^{\otimes n} \otimes I^{\otimes n}) U_f (H^{\otimes n} \otimes I^{\otimes n})(|\vec{0}\rangle_n \otimes |\vec{0}\rangle_n).
\end{aligned}$$

For $|\psi\rangle$, we know that $H^{\otimes n}$ creates a uniform superposition of $|\vec{j}\rangle_n, \vec{j} \in \{0,1\}^n$ over the first n quantum bits. Therefore, we can write

$$|\psi\rangle = (H^{\otimes n} \otimes I^{\otimes n})(|\vec{0}\rangle_n \otimes |\vec{0}\rangle_n) = \frac{1}{\sqrt{2^n}} \sum_{\vec{j}\in\{0,1\}^n} |\vec{j}\rangle_n \otimes |\vec{0}\rangle_n.$$

By linearity, applying U_f to this state yields

$$|\phi\rangle = U_f|\psi\rangle = \frac{1}{\sqrt{2^n}} \sum_{\vec{j}\in\{0,1\}^n} |\vec{j}\rangle_n \otimes |\vec{0} \oplus f(\vec{j})\rangle_n = \frac{1}{\sqrt{2^n}} \sum_{\vec{j}\in\{0,1\}^n} |\vec{j}\rangle_n \otimes |f(\vec{j})\rangle_n.$$

We now need to analyze the effect of applying further Hadamard gates on the top lines of the circuit. Using (3.1), the next step in the circuit is given by

$$\begin{aligned} |\chi\rangle &= (H^{\otimes n} \otimes I^{\otimes n}) \frac{1}{\sqrt{2^n}} \sum_{\vec{j}\in\{0,1\}^n} |\vec{j}\rangle_n \otimes |f(\vec{j})\rangle_n \\ &= \frac{1}{2^n} \sum_{\vec{j}\in\{0,1\}^n} \sum_{\vec{k}\in\{0,1\}^n} (-1)^{\vec{k}\bullet\vec{j}} |\vec{k}\rangle_n \otimes |f(\vec{j})\rangle_n. \end{aligned} \tag{4.1}$$

When we make a measurement on the top n qubit lines of $|\chi\rangle$ (i.e., the first n-qubit register, containing qubits 1 through n), we obtain one of the strings $\vec{k}$ with probability equal to the sum of the modulus squared of the coefficient of the states $|\vec{k}\rangle_n \otimes |f(\vec{j})\rangle$ for all $\vec{j}$. It is easier to analyze the case $\vec{a} \neq \vec{0}$ first; we will deal with the case $\vec{a} = 0$ later in section 4.3. Assuming $\vec{a} \neq \vec{0}$, $|\vec{k}\rangle_n \otimes |f(\vec{j})\rangle_n = |\vec{k}\rangle_n \otimes |f(\vec{j} \oplus \vec{a})\rangle_n$. Let R be a set with the following property: for every $\vec{j} \in \{0,1\}^n$, R contains either $\vec{j}$ or $\vec{j} \oplus \vec{a}$, but not both. (For the reader familiar with the concept of quotient sets, R is the quotient set $\{0,1\}^n/\sim$, where $\sim$ is the equivalence relationship defined as $\vec{x} \sim \vec{y}$ if and only if $\vec{x} = \vec{y} \oplus \vec{a}$.) Then, for each $\vec{k}$, the string $\vec{k}$ appears in the top qubit lines exactly in the 2^{n-1} basis states $|\vec{k}\rangle_n \otimes |f(\vec{j})\rangle_n$ for $\vec{j} \in R$. For each $\vec{j} \in R$, the coefficient of the basis state $|\vec{k}\rangle_n \otimes |f(\vec{j})\rangle_n$ is exactly the sum of the coefficients in (4.1) for $|\vec{k}\rangle_n \otimes |f(\vec{j})\rangle_n$ and $|\vec{k}\rangle_n \otimes |f(\vec{j} \oplus \vec{a})\rangle_n$; that is,

$$\begin{aligned} \frac{(-1)^{\vec{k}\bullet\vec{j}} + (-1)^{\vec{k}\bullet(\vec{j}\oplus\vec{a})}}{2^n} &= \frac{(-1)^{\vec{k}\bullet\vec{j}} + (-1)^{\vec{k}\bullet\vec{j}}(-1)^{\vec{k}\bullet\vec{a}}}{2^n} \\ &= \frac{(-1)^{\vec{k}\bullet\vec{j}} \left(1 + (-1)^{\vec{k}\bullet\vec{a}}\right)}{2^n}. \end{aligned}$$

Therefore, the probability of obtaining the binary string $\vec{k}$ after measuring the top qubit lines is

$$\begin{aligned} \sum_{\vec{j}\in R} \left(\frac{(-1)^{\vec{k}\bullet\vec{j}} \left(1 + (-1)^{\vec{k}\bullet\vec{a}}\right)}{2^n} \right)^2 &= 2^{n-1} \left(\frac{\left(1 + (-1)^{\vec{k}\bullet\vec{a}}\right)}{2^n} \right)^2, \\ &= \begin{cases} \frac{1}{2^{n-1}} & \text{if } \vec{k} \bullet \vec{a} \equiv 0 \mod 2, \\ 0 & \text{if } \vec{k} \bullet \vec{a} \equiv 1 \mod 2, \end{cases} \end{aligned}$$

where the multiplication factor 2^{n-1} comes from the fact that $|R| = \frac{2^n}{2}$. Thus, the only binary strings that have positive probability to be observed are those strings $\vec{k}$ for which $\vec{k} \bullet \vec{a} \equiv 0 \mod 2$. The remaining strings are never sampled: by carefully applying quantum operations we have reduced their state coefficients to zero, a phenomenon known as *destructive interference*. Notice that unless $\vec{k} = \vec{0}$, there is a nonempty set of bits for which the modulo 2 sum of $\vec{a}$ must vanish. In this case, unless we are unlucky and we obtain the vector $\vec{k} = \vec{0}$ (or some other undesirable cases that will be specified later), we can express one such bit as a modulo 2 sum of the others, and we eliminate half of the possible values for $\vec{a}$.

Our discussion shows that with a single quantum query to U_f, in the case $\vec{a} \neq \vec{0}$ with high probability we learn very valuable information about $\vec{a}$, and we can approximately halve the search space for $\vec{a}$. It now remains to fully specify in a more precise manner how this information can be used, and how to deal with the case $\vec{a} = \vec{0}$.

4.3. Simon's Algorithm: Description and Analysis. The quantum algorithm described in the previous section yields information on $\vec{a}$, but it does not output $\vec{a}$ directly. To recover $\vec{a}$, further calculations have to be performed. This is a typical situation in quantum algorithms: a quantum computation measures some properties of the desired answer, then classical computations are used to analyze these properties and obtain the desired answer. Thus, even if the quantum algorithm does not explicitly output the desired answer, it allows us to move closer to our goal.

In the specific case of the problem discussed here, the quantum computation allows us to learn $\vec{k}$ such that $\vec{k} \bullet \vec{a} \equiv 0 \mod 2$. We embed this equation into an algorithm as follows: we initialize E to the empty set, and then, while the system of equations E does not have a unique solution, we apply the circuit described in the previous section to obtain $\vec{k}$ and add the equation $\vec{k} \bullet \vec{a} \equiv 0 \mod 2$ to E. Notice that $\vec{a} = \vec{0}$ is always a solution of the homogeneous system E, but we are interested in the nonzero solutions, if any exist. In other words, we want to determine whether the null space contains any nonzero vector. We can have two possible situations: either the system has a uniquely determined nonzero solution $\vec{a} \neq \vec{0}$, or the only possible solution is $\vec{a} = \vec{0}$. Since there are n unknowns and we are dealing with a homogeneous system, to identify which of these situations occurs we need E to contain n linearly independent vectors $\vec{k}$, where independence is intended modulo 2. Because at every iteration we obtain a random $\vec{k}$ for which $\vec{k} \bullet \vec{a} \equiv 0 \mod 2$, we need to analyze how many iterations we need to obtain n such vectors with high probability.

In continuous space, uniform random sampling of vectors yields linearly independent vectors with probability 1. In this case, we are considering linear independence among vectors that have coefficients 0 or 1, and independence is in terms of the modulo 2 sum, so the argument is less clear; however, it is possible to show that the probability of obtaining n such linearly independent vectors after sampling $n + t$ times is bounded below by $1 - \frac{1}{2^t}$ [29, App. G]. This lower bound does not depend on n. Hence, with overwhelming probability after slightly more than n executions of the quantum circuit, and therefore $O(n)$ queries to the function f, we determine the solution to the problem with a classical computation that can be performed in polynomial time (i.e., $O(n^2)$ to determine a solution to the system of linear equations modulo 2). We remark that once the unique nonzero $\vec{a}$ is determined, we can easily verify that it is the solution by querying the function. On the other hand, if $\vec{a} = \vec{0}$, the algorithm will detect that this is the case because at some point the system of linear equations E will have $\vec{a} = \vec{0}$ as the only possible solution (notice that if $\vec{a} = \vec{0}$, then $\vec{k} \bullet \vec{a} \equiv 0 \mod 2$ for any $\vec{k}$). Compare the $O(n)$ queries of this approach with the

$O(2^{n/2})$ queries that are required by a classical algorithm, and we have just shown an exponential speedup.

This algorithm shows a typical feature of many quantum algorithms: often, there is a classical computation to complement the quantum computation. For example, the classical computation could verify that the correct solution to the problem has indeed been found. In this case, the verification is carried out by checking whether the system of equations has a unique solution. Indeed, quantum algorithms are probabilistic algorithms, and we can only try to increase the probability that the correct answer is returned; only in rare cases can the solution be obtained with probability 1; see, e.g., [10]. For this reason, it is desirable to have a way to deterministically verify correctness. This may require a classical computation. In other words, the quantum algorithm is applied to a problem for which it is difficult to classically compute the solution, but once the solution (or some information about it) is obtained, it is easy to classically verify that we have the right answer. This is not known to be possible in general, since the complexity class BQP is not known or believed to be contained in NP (recall that NP is the class of problems admitting an efficient classical proof that the solution has been found, i.e., a certificate that can be checked in polynomial time); indeed, it is an active topic of research to design verification protocols for generic quantum computations [11, 1, 31, 28]. However, the quantum algorithms presented in this paper will admit simple classical verification.

5. Black-Box Search: Grover's Algorithm. Simon's algorithm gives an exponential speedup with respect to a classical algorithm, but it solves a very narrow problem that does not have practical applications. We now describe an algorithm that gives only a polynomial—more specifically, quadratic—speedup with respect to classical algorithms, but it applies to a very large class of problems. This algorithm is known as Grover's search [21].

The problem solved by the algorithm can be described as black-box search: we are given a circuit that computes an unknown function of a binary string, and we want to determine for which value of the input the function gives output 1. In other words, we are trying to determine the unique binary string that satisfies a property encoded by a circuit. The original paper [21] describes this as looking for a certain element in a database. Such an algorithm can be applied whenever we are searching for a specific element in a set, we have a way of testing whether an element is the desired element (in fact, this test must be implementable as a quantum subroutine—see below), and we do not have enough information to do anything smarter than a brute force search, i.e., testing all elements in the set.

As mentioned earlier, the basic idea of the algorithm is to start with the uniform superposition of all basis states and iteratively increase the coefficients of basis states that correspond to binary strings for which the unknown function gives output 1.

We need some definitions. Let $f : \{0,1\}^n \to \{0,1\}$ and assume that there exists a unique $\vec{\ell} \in \{0,1\}^n : f(\vec{\ell}) = 1$, i.e., there is a unique element in the domain of the function that yields output 1. We want to determine $\vec{\ell}$. The function f is assumed to be encoded by a unitary as follows:

$$U_f : |\vec{\jmath}\rangle_n \otimes |y\rangle_1 \to |\vec{\jmath}\rangle_n \otimes |y \oplus f(\vec{\jmath})\rangle_1.$$

As usual, we are allowed to query the function in superposition.

Remark 5.1. Grover's search can also be applied to the case in which there are multiple input values that yield output 1 and we want to retrieve any of them; however,

the analysis in that case is slightly more convoluted and is not pursued here in detail. By the end of our analysis, the reader will have all the necessary tools to study this extension.

5.1. Classical Algorithm. Given the problem definition, classical search cannot do better than $O(2^n)$ operations. Indeed, any deterministic classical algorithm may need to explore all 2^n possible input values before finding $\vec{\ell}$: given any deterministic classical algorithm, there exists a permutation π of $\{0,1\}^n$ that represents the longest execution path (i.e., sequence of values at which f is queried) of such an algorithm. Then, if $\vec{\ell} = \pi(\vec{1})$, i.e., the last element of the longest execution path, the algorithm will require 2^n queries to determine the answer, which is clearly the worst case.

At the same time, a randomized algorithm requires $O(2^n)$ function calls to have at least a constant positive probability of determining $\vec{\ell}$; the expected number of function calls to determine the answer is approximately 2^{n-1}, i.e., the expected number of draws before we extract the black marble from an urn containing one black marble and $2^n - 1$ white marbles (sampling without replacement; the corresponding distribution is known as the hypergeometric distribution).

5.2. Grover's Search: Algorithm Description. The quantum search algorithm proposed in [21] requires $q = n + 1$ qubits, which is exactly the number of qubits for the encoding of U_f as defined above.

The outline of the algorithm is as follows. The algorithm starts with the uniform superposition of all basis states on n qubits. The last qubit $(n + 1)$ is used as an auxiliary qubit, and it is initialized to $H|1\rangle$. We obtain the quantum state $|\psi\rangle$. Then, these operations are repeated several times:

(i) Flip the sign of the vectors for which U_f gives output 1.
(ii) Invert all the coefficients of the quantum state around the average coefficient—we will explain the precise mapping implemented by this operation in section 5.2.3.

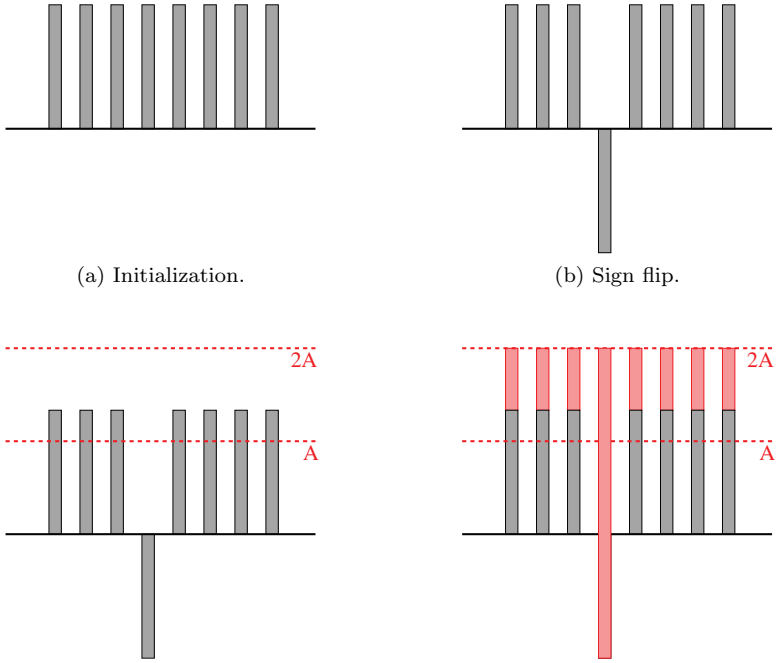
A full cycle of the two operations above increases the coefficient of $|\vec{\ell}\rangle_n \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$, and after a certain number of cycles (to be specified later), the coefficient of the state $|\vec{\ell}\rangle_n \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$ is large enough that it can be obtained from a measurement with probability close to 1. This phenomenon is known as *amplitude amplification*.

A sketch of the ideas for the algorithm is depicted in Figure 16: we have eight basis states, and suppose the fourth basis state is the target basis state $|\vec{\ell}\rangle$. The representation is purely meant to convey intuition and does not geometrically represent the vectors encoding the quantum state, but solely the amplitude of the coefficients. In Figure 16(a), all basis states have the same coefficient. In Figure 16(b), the coefficient of the target basis state has its sign flipped. In Figure 16(c), we can see that the average value of the coefficients is slightly below the coefficient for the undesired states. Taking twice the average and subtracting each coefficient now yields the red bars in Figure 16(d), where the target basis state $|\vec{\ell}\rangle$ has a coefficient with much larger value than the rest and will therefore be measured with higher probability. Of course, we need to show that these steps can be implemented with unitary matrices that can be constructed with a polynomial number of basic gates.

We now describe the steps above in more detail.

5.2.1. Initialization. The algorithm is initialized by applying the operation $H^{\otimes(n+1)}(I^{\otimes n} \otimes X)$ onto the state $|\vec{0}\rangle_{n+1}$. We can express the quantum state as

$$(I^{\otimes n} \otimes X)|\vec{0}\rangle_{n+1} = |\vec{0}\rangle_n \otimes |1\rangle,$$



(a) Initialization. (b) Sign flip.

(c) Computation of the average. (d) Inversion about the average.

Fig. 16 *Sketch of Grover's algorithm. The bars represent the coefficients of the basis states.*

$$H^{\otimes(n+1)}(I^{\otimes n}\otimes X)|\vec{0}\rangle_{n+1} = \sum_{\vec{\jmath}\in\{0,1\}^n} \frac{1}{\sqrt{2^n}}|\vec{\jmath}\rangle_n \otimes \frac{(|0\rangle - |1\rangle)}{\sqrt{2}}$$
$$= \sum_{\vec{\jmath}\in\{0,1\}^n} \alpha_{\vec{\jmath}}|\vec{\jmath}\rangle_n \otimes \frac{(|0\rangle - |1\rangle)}{\sqrt{2}} = |\psi\rangle,$$

where $\alpha_{\vec{\jmath}} = \frac{1}{\sqrt{2^n}}$. Thus, the initial coefficients $\alpha_{\vec{\jmath}}$ of the state $|\psi\rangle$ are real numbers. Since all the other steps of the algorithm will map real numbers to real numbers, we only need to consider real numbers throughout the algorithm.

5.2.2. Sign Flip: Step (i). To flip the sign of the target state $|\vec{\ell}\rangle_n \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$, we apply U_f to $|\psi\rangle$. We now show why this works:

$$U_f|\psi\rangle = U_f\left(\sum_{\vec{\jmath}\in\{0,1\}^n} \alpha_{\vec{\jmath}}|\vec{\jmath}\rangle_n \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)\right)$$
$$= \alpha_{\vec{\ell}}|\vec{\ell}\rangle_n \otimes \frac{1}{\sqrt{2}}(|1\rangle - |0\rangle) + \sum_{\substack{\vec{\jmath}\in\{0,1\}^n \\ \vec{\jmath}\neq\vec{\ell}}} \alpha_{\vec{\jmath}}|\vec{\jmath}\rangle_n \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$
$$= \left(-\alpha_{\vec{\ell}}|\vec{\ell}\rangle_n + \sum_{\substack{\vec{\jmath}\in\{0,1\}^n \\ \vec{\jmath}\neq\vec{\ell}}} \alpha_{\vec{\jmath}}|\vec{\jmath}\rangle_n\right) \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle).$$

As the expression above suggests, we can always think of the last qubit as being in the state $\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$ and unentangled from the rest of the qubits, with the sign flip affecting only the first n qubits. Therefore, the state that we obtain by applying U_f to $|\psi\rangle$ is the same as $|\psi\rangle$, except that the sign of $|\vec{\ell}\rangle_n \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$ has been flipped.

5.2.3. Inversion about the Average: Step (ii). To perform the inversion about the average, we want to perform the operation

$$\sum_{\vec{\jmath}\in\{0,1\}^n} \alpha_{\vec{\jmath}} |\vec{\jmath}\rangle_n \to \sum_{\vec{\jmath}\in\{0,1\}^n} \left(2 \left(\sum_{\vec{k}\in\{0,1\}^n} \frac{\alpha_{\vec{k}}}{2^n} \right) - \alpha_{\vec{\jmath}} \right) |\vec{\jmath}\rangle_n,$$

where $\sum_{\vec{k}\in\{0,1\}^n} \frac{\alpha_{\vec{k}}}{2^n}$ is the average, and therefore we are taking twice the average and subtracting each coefficient from it. It is not clear yet that this is a unitary operation, but it will become evident in what follows. This mapping is realized by the matrix

$$W = \begin{pmatrix} \frac{2}{2^n} - 1 & \frac{2}{2^n} & \cdots & \frac{2}{2^n} \\ \frac{2}{2^n} & \frac{2}{2^n} - 1 & \cdots & \frac{2}{2^n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{2}{2^n} & \frac{2}{2^n} & \cdots & \frac{2}{2^n} - 1 \end{pmatrix} = \begin{pmatrix} \frac{2}{2^n} & \frac{2}{2^n} & \cdots & \frac{2}{2^n} \\ \frac{2}{2^n} & \frac{2}{2^n} & \cdots & \frac{2}{2^n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{2}{2^n} & \frac{2}{2^n} & \cdots & \frac{2}{2^n} \end{pmatrix} - I^{\otimes n},$$

where the denominator $\frac{1}{2^n}$ computes the average coefficient, the numerator 2 of the fraction takes twice the average, and finally we subtract the identity to subtract each individual coefficient from twice the average. From the definition of the Hadamard gate in (3.1), we can see that the entry of $H^{\otimes n}$ in position j, k is $(H^{\otimes n})_{j,k} = \frac{1}{\sqrt{2^n}}(-1)^{\vec{\jmath}\bullet\vec{k}}$. However, if we now let

$$R = \begin{pmatrix} 2 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \dots & 0 \end{pmatrix} \in \mathbb{R}^{2^n \times 2^n},$$

then we can write $(H^{\otimes n} R H^{\otimes n})_{j,k} = (H^{\otimes n})_{j,0} R_{0,0} (H^{\otimes n})_{0,k} = \frac{2}{2^n}$, because $R_{j,k} = 0$ for $j \neq 0$ or $k \neq 0$. Therefore, using the fact that $H^{\otimes n} H^{\otimes n} = I^{\otimes n}$, we have

$$\begin{aligned} W &= H^{\otimes n} R H^{\otimes n} - I^{\otimes n} = H^{\otimes n}(R - I^{\otimes n}) H^{\otimes n} = -H^{\otimes n}(I^{\otimes n} - R) H^{\otimes n} \\ &= -H^{\otimes n} \mathrm{diag}(\underbrace{-1, 1, \dots, 1}_{2^n}) H^{\otimes n} := -H^{\otimes n} D H^{\otimes n}. \end{aligned} \tag{5.1}$$

The expression (5.1), besides providing a decomposition for W, also shows that W is unitary, because $H^{\otimes n}$ is unitary (tensor product of unitary matrices) and D is diagonal with ones on the diagonal. We must find a way to construct the matrix $D := \mathrm{diag}(-1, 1, \dots, 1)$. This will be discussed in the next section. For now, we summarize our analysis of the inversion about the average by concluding that it can be performed by applying $W = -H^{\otimes n} D H^{\otimes n}$ to the n qubits of interest (i.e., all qubits except the auxiliary qubit that we used for the sign flip of step (i)).

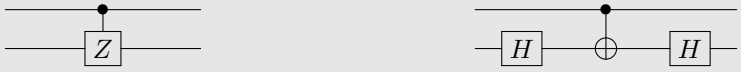


Fig. 17 *Controlled Z gate on two qubits: two possible representations.*

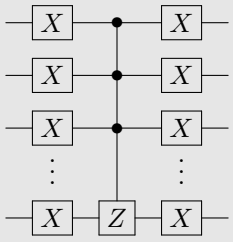


Fig. 18 *Quantum circuit implementing the D operation used in Grover's algorithm.*

5.2.4. Constructing the Matrix D. We give a sketch of the idea of how to construct $D = \mathrm{diag}(-1, 1, \ldots, 1)$. Notice that the effect of this quantum operation is to flip the sign of the coefficient of the basis state $|\vec{0}\rangle_n$ and leave other coefficients untouched.

Instead of flipping the sign of $|\vec{0}\rangle_n$, let us start by seeing how to flip the sign of $|\vec{1}\rangle_n$ while leaving all other coefficients untouched. Let $\mathrm{C}^{n-1}Z$ be the gate that applies Z to qubit n if qubits $1, \ldots, n-1$ are $|1\rangle$, and does nothing otherwise. This is similar to the CNOT gate, except that it has multiple controls, and it applies a Z gate rather than an X (i.e., NOT) gate when the control qubits are $|1\rangle$. It is called a "multiply-controlled Z." $\mathrm{C}^{n-1}Z$ in the case of two qubits ($n = 2$) is given by the following matrix:

$$\mathrm{C}Z = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}.$$

Notice that in the 2-qubit case ($n = 2$), the two circuits depicted in Figure 17 are equivalent: carrying out the matrix multiplications will confirm that the circuit on the right in Figure 17 implements exactly the $\mathrm{C}Z$ matrix as defined above. Thus, the controlled Z gate can be easily realized with a CNOT and two Hadamard gates. If we have access to the $\mathrm{C}^{n-1}Z$ gate, we can write

$$D = X^{\otimes n}(\mathrm{C}^{n-1}Z)X^{\otimes n},$$

because, as can be easily verified, this operations flips the sign of the coefficient of a basis state if and only if all qubits have value $|0\rangle$ in the basis state. In circuit form, it can be written as depicted in Figure 18.

Of course, one has to construct the operation $\mathrm{C}^{n-1}Z$. There are several ways to do so. Perhaps the simplest construction, suggested in [6], is to implement a C^{n-2}NOT and a controlled Z gate. The C^{n-2}NOT is actually easy to implement with some

auxiliary qubits, in a construction that will be used in section 6 as well. We show this scheme in Figure 19 with an example for $n = 4$ qubits, but clearly it can be generalized to an arbitrary number of qubits. We first implement a $\mathrm{C}^{n-2}\mathrm{NOT}$ gate, with an auxiliary qubit (which is initialized to $|0\rangle$, as one can see from the bottom qubit in Figure 19) as the target of the $\mathrm{C}^{n-2}\mathrm{NOT}$. We then implement a $\mathrm{CC}Z$ gate using a CCNOT and two Hadamard gates on the target qubit; readers can easily convince themselves that this implements a doubly controlled Z, using the identity $HXH = Z$ and carrying out the calculations (in the large unitary matrix for $\mathrm{CC}Z$, the gate being controlled appears in the bottom right, just as in CNOT). Summarizing, this yields a decomposition of $\mathrm{C}^{n-1}Z$ with a linear number of gates and auxiliary qubits. It is possible to forsake the initialization of the auxiliary qubit; see [6] for details. To conclude, the construction of D, and therefore of the whole circuit implementing step (ii) of Grover's search, can be performed in $O(n)$ gates and auxiliary qubits. We remark that the -1 multiplication factor appearing in front of $H^{\otimes n}$ in (5.1) can be ignored, as it is a global phase factor; see Example 3.9.

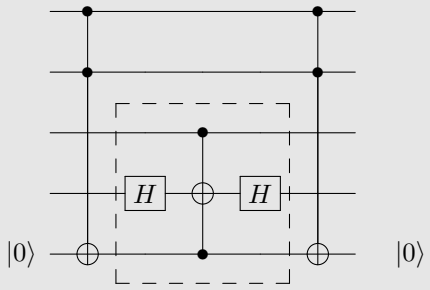


Fig. 19 *Decomposition of $C^{n-1}Z$ for $n = 4$. The fifth (bottom) qubit is initialized to $|0\rangle$ and is used as working space. This implements C^3Z for the top four qubits.*

5.3. Determining the Number of Iterations. Let Q be the matrix that applies a single iteration of Grover's search, consisting of steps (i) and (ii) above. It is paramount to determine how many iterations should be performed, so that the coefficient of the desired basis state $|\vec{\ell}\rangle \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$ is as large as possible and the binary string $\vec{\ell}$ is the outcome of a measurement with high probability. This is what we attempt to do in this section.

Since the last, auxiliary qubit is always in state $\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$ and unentangled with the rest, we can ignore it. Let

$$|\psi_D\rangle = |\vec{\ell}\rangle_n, \qquad |\psi_U\rangle = \left(\sum_{\substack{\vec{\jmath} \in \{0,1\}^n \\ \vec{\jmath} \neq \vec{\ell}}} \frac{1}{\sqrt{2^n - 1}} |\vec{\jmath}\rangle_n \right)$$

be the desirable and undesirable quantum states, respectively. We claim that after iteration k of the algorithm, the quantum state can be expressed as $|\psi_k\rangle = d_k|\psi_D\rangle + u_k|\psi_U\rangle$. We will show this by induction. Initially, $d_0 = \frac{1}{\sqrt{2^n}}$ and $u_0 = \sqrt{\frac{2^n - 1}{2^n}}$, where notice that to obtain u_0 from the value of an individual coefficient in $|\psi_U\rangle$ (all such

coefficients are $\frac{1}{\sqrt{2^n}}$ initially) we have multiplied by $\sqrt{2^n-1}$ for normalization. Thus, the claim is true for $k=0$. We now need to show the induction step: assuming $|\psi_{k-1}\rangle = d_{k-1}|\psi_D\rangle + u_{k-1}|\psi_U\rangle$, we must show $|\psi_k\rangle = d_k|\psi_D\rangle + u_k|\psi_U\rangle$.

The calculations in this part are heavier than usual; if the reader is not interested in the details, they can simply trust the results and skip to the end of this gray box.

At step (i) of the algorithm, the algorithm flips $d_k|\psi_D\rangle + u_k|\psi_U\rangle \to -d_k|\psi_D\rangle + u_k|\psi_U\rangle$.

At step (ii), the algorithm maps $\alpha_h \to 2A_k - \alpha_h$ for each coefficient α_h, where A_k is the average coefficient. Therefore,

$$\begin{aligned} -\alpha_{\vec{\ell}} &\to 2A_k + \alpha_{\vec{\ell}}, \\ \alpha_{\vec{h}} &\to 2A_k - \alpha_{\vec{h}} \quad \forall \vec{h} \neq \vec{\ell}. \end{aligned}$$

To compute A_k, we need to determine the value of each individual coefficient. The coefficient for $|\vec{\ell}\rangle$ is clearly d_k, as there is only one such state. On the other hand, there are 2^n-1 states with coefficient u_k, so the value of the coefficient for each of the states $|\vec{\jmath}\rangle, \vec{\jmath} \neq \vec{\ell}$ is $\frac{u_k}{\sqrt{2^n-1}}$ (the square root is due to normalization; see above). The average coefficient at iteration k is therefore

$$A_k = \frac{(2^n-1)\frac{1}{\sqrt{2^n-1}}u_k - d_k}{2^n} = \frac{\sqrt{2^n-1}u_k - d_k}{2^n}.$$

To obtain u_k from $\alpha_{\vec{h}}$ we need to multiply by $\sqrt{2^n-1}$, so the mapping of step (ii) can be written, overall, as

$$\begin{aligned} -d_k|\psi_D\rangle + u_k|\psi_U\rangle \to &(2A_k + d_k)|\psi_D\rangle + \sqrt{2^n-1}\left(2A_k - \frac{u_k}{\sqrt{2^n-1}}\right)|\psi_U\rangle \\ &= d_{k+1}|\psi_D\rangle + u_{k+1}|\psi_U\rangle, \end{aligned}$$

where we have defined

$$\begin{aligned} d_{k+1} &= 2A_k + d_k, \\ u_{k+1} &= 2A_k\sqrt{2^n-1} - u_k. \end{aligned}$$

This shows the induction step.

Performing the substitution of A_k, we obtain

$$d_{k+1} = 2\frac{\sqrt{2^n-1}u_k - d_k}{2^n} + d_k = \left(1 - \frac{1}{2^{n-1}}\right)d_k + \frac{2\sqrt{2^n-1}}{2^n}u_k,$$

$$u_{k+1} = 2\frac{\sqrt{2^n-1}u_k - d_k}{2^n}\sqrt{2^n-1} - u_k = -\frac{2\sqrt{2^n-1}}{2^n}d_k + \left(1 - \frac{1}{2^{n-1}}\right)u_k.$$

This transformation is exactly a clockwise rotation of the vector $\binom{d_k}{u_k}$ by a certain angle θ, because it has the form

$$\begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}\begin{pmatrix} d_k \\ u_k \end{pmatrix}$$

and it satisfies the relationship $\sin^2\theta + \cos^2\theta = 1$. The angle θ must satisfy

$$\sin\theta = \frac{2\sqrt{2^n - 1}}{2^n}. \tag{5.2}$$

Notice that because this value of the sine is very small (for large n), we can use the approximation $\sin x \approx x$ (when x is close to 0) to write

$$\theta = \frac{2\sqrt{2^n - 1}}{2^n}. \tag{5.3}$$

Overall, the above analysis shows that each iteration performs a rotation by an angle θ of the vector $|\psi_k\rangle$, which always belongs to the plane spanned by $|\psi_D\rangle$ and $|\psi_U\rangle$. So after k iterations the coefficients d_k, u_k satisfy the equation

$$\begin{pmatrix} d_k \\ u_k \end{pmatrix} = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}^k \begin{pmatrix} d_0 \\ u_0 \end{pmatrix},$$

which can be rewritten as

$$\begin{aligned} d_k &= \cos k\theta d_0 + \sin k\theta u_0, \\ u_k &= -\sin k\theta d_0 + \cos k\theta u_0. \end{aligned}$$

In order to maximize the probability of obtaining $|\psi_D\rangle$ after a measurement, remember that $|u_0| \gg |d_0|$, so the best choice is to pick $k\theta = \frac{\pi}{2}$, which yields the largest value of $|d_k|$. Using (5.3), the optimal number of iterations of Grover's search algorithm is

$$k \approx \frac{2^n\pi}{4\sqrt{2^n - 1}} \approx \frac{\pi}{4}\sqrt{2^n}. \tag{5.4}$$

After this many iterations, we have a probability close to 1 of measuring $|\psi_D\rangle$ and obtaining the sought state $|\vec{\ell}\rangle$. Comparing this with a classical algorithm, which requires $O(2^n)$ iterations, we have obtained a quadratic speedup.

Remark 5.2. If we perform more iterations of Grover's algorithm than the optimal number, the probability of measuring the desired state actually goes down, and this reduces our chances of success. Therefore, it is important to choose the right number of iterations.

Of course, the approximation for θ given in (5.3) is only valid for large n; for smaller n, it is better to compute the optimal number of iterations deriving θ from (5.2). We conclude this section by noticing that in the case that there are multiple input values on which f has value 1, we should amend the above analysis by adjusting the values for d_0 and u_0, but the main steps remain the same.

6. Numerical Implementation of Grover's Algorithm.

We conclude the paper by showing how to implement Grover's algorithm using the open-source Python library Qiskit [4]. One of the advantages of using Qiskit is that the code is self-explanatory. The library requires Python 3 and can be installed via `pip`:

```
pip install qiskit==0.11.1
```

The code below was tested with Qiskit 0.11.1; it may work with other versions as well.

We apply Grover's algorithm to the problem of finding a satisfying assignment for an instance of the Exactly-1 3-SAT problem, defined as follows:

Problem (Exactly-1 3-SAT): Determining a satisfying assignment containing one true literal per clause.
Input: SAT formula in conjunctive normal form $\bigwedge_{k=1}^{m} C_k$ over n Boolean variables $x_1, \ldots, x_n$, with 3 literals per clause $C_1, \ldots, C_m$.
Output: Does there exist an assignment of $x_1, \ldots, x_n$ such that every clause $C_1, \ldots, C_m$ has exactly one `True` literal?

This problem is NP-hard [20]. In our implementation, an instance of Exactly-1 3-SAT is specified as a list of clauses, where each clause contains three integers: a positive integer is the index of a positive literal, and a negative integer is the index of a negative literal. For example, the Python list of lists

$$[[1, 2, -3], [-1, -2, -3], [-1, 2, 3]]$$

represents the instance

$$(x_1 \bigtriangledown x_2 \bigtriangledown \neg x_3) \wedge (\neg x_1 \bigtriangledown \neg x_2 \bigtriangledown \neg x_3) \wedge (\neg x_1 \bigtriangledown x_2 \bigtriangledown x_3). \tag{6.1}$$

We use this formula in the example given below. We use the symbol $\bigtriangledown$ rather than the usual $\vee$ (normally used to indicate the logical OR) to emphasize that this is not a regular 3-SAT formula, but an Exactly-1 3-SAT formula: the problem definition requires *exactly* one True literal per clause.

In the implementation presented in this section we only allow up to three Boolean variables and three clauses, i.e., $n \leq 3, m \leq 3$. At the end of the section we will discuss how to generalize the implementation to allow an arbitrary number of clauses, which is left as an exercise. Notice that the suggested approach requires additional qubits for each clause. The code presented here for $n \leq 3, m \leq 3$ yields a circuit with at most 8 qubits, which can be simulated on a laptop in less than one minute in most cases. Further clauses would require additional qubits and slow down the simulation of the circuit (roughly by a factor of 2 for each additional qubit).

To apply Grover's algorithm we will use three basic subroutines: a subroutine to construct the initial state, a subroutine to compute the unitary U_f implementing the black-box function f, and a subroutine to perform the inversion about the average. We will discuss them in order.

6.1. Initial State. Before we construct the initial state, let us give names to some of the quantum registers (i.e., groups of quantum qubits) that we need. Grover's algorithm applies to a function with an n-qubit input and a single-qubit output. We call `f_in` the input register of U_f, of size n, and `f_out` the output register of U_f, of size 1. We construct the initial state as follows:

```
def input_state(circuit, f_in, f_out, n):
    """(n+1)-qubit input state for Grover search."""
    for j in range(n):
        circuit.h(f_in[j])
    circuit.x(f_out)
    circuit.h(f_out)
```

This is equivalent to the circuit given in Figure 20.

6.2. Black-Box Function U_f. Implementing U_f for the Exactly-1 3-SAT problem is the most complex part of the code, and it can be done in several ways. To favor

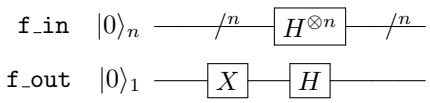


Fig. 20 *Initialization step of Grover's algorithm.*

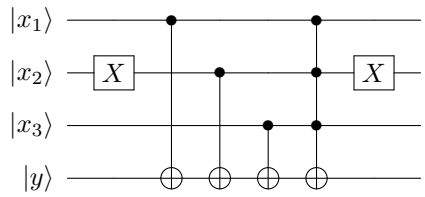


Fig. 21 *Quantum circuit bit-flipping the bottom qubit if the clause $x_1 \bigtriangledown \neg x_2 \bigtriangledown x_3$ is satisfied by exactly one literal.*

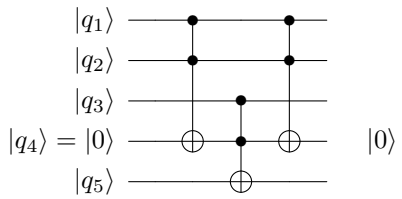


Fig. 22 *A possible implementation of a triply controlled NOT gate.*

simplicity, we decompose the problem of computing U_f by introducing m auxiliary qubits (these are often called "ancillas" in the quantum computing literature), one for each clause. For each clause we construct a circuit that bit-flips the corresponding auxiliary qubit if and only if the clause has exactly one true literal (these auxiliary qubits will be initialized in state $|0\rangle$). Finally, we apply a bit flip on the output register of U_f if and only if all m auxiliary qubits are 1. In Figure 21 we show a circuit that bit-flips the bottom qubit y if the clause $x_1 \bigtriangledown \neg x_2 \bigtriangledown x_3$ is satisfied. The idea is as follows. The X gate bit-flips the qubit $|x_2\rangle$, since x_2 appears negated in the clause. Using three CNOT gates, we set $|y\rangle = |y \oplus x_1 \oplus \neg x_2 \oplus x_3\rangle$, implying that y is bit-flipped if an odd number of literals is satisfied. We use a triply controlled NOT gate to finally obtain $|y\rangle = |y \oplus x_1 \oplus \neg x_2 \oplus x_3 \oplus (x_1 \wedge \neg x_2 \wedge x_3)\rangle$, as desired. To set $|y\rangle = |1\rangle$ if and only if exactly one literal is satisfied, it is enough to assume that $|y\rangle$ starts in $|0\rangle$ state. The final X gate resets the state of qubit $|x_2\rangle$.

To implement this circuit in Qiskit, there is a small obstacle: the triply controlled NOT gate is not part of the basic gate set. We can implement it with a strategy similar to what is discussed in section 5.2.4 for the $\mathrm{C}^{n-1}Z$ gate: we use three CCNOT gates and one auxiliary qubit, implementing the circuit in Figure 22. This has the drawback of requiring an extra qubit, but it is easy to understand. We remark that CCNOT, while not part of the basic gate set of Qiskit, is defined as a macro and therefore can be used as if it were part of the basic gate set: the software will automatically perform the substitution, using the circuit of Figure 13 to implement CCNOT.

We can quickly verify that the circuit in Figure 22 bit-flips $|q_5\rangle$ if and only if $|q_1\rangle, |q_2\rangle, |q_3\rangle$ are 1; the final CCNOT resets the state of the auxiliary qubit $|q_4\rangle$, which is assumed to be initialized at $|0\rangle$ and is left in state $|0\rangle$.

The implementation of U_f then proceeds as follows. We loop over the clauses, using index $k = 0, \ldots, m-1$ (as often happens, in this paper we use "mathematical

language" and the clauses are numbered $k = 1, \ldots, m$, but in the Python implementation the corresponding array is indexed $k = 0, \ldots, m-1$). For each clause we implement the circuit in Figure 21, setting the auxiliary qubit `aux[k]` to 1 if clause C_{k+1} is satisfied. We then perform a multiply-controlled NOT operation to ensure that the output register `f_out` is bit-flipped if all m auxiliary qubits are 1. Finally, we run the same circuit in reverse to reset the state of the auxiliary qubits.

```
def black_box_u_f(circuit, f_in, f_out, aux, n, exactly_1_3_sat_formula):
    """Circuit that computes the black-box function from f_in to f_out.

    Create a circuit that verifies whether a given exactly-1 3-SAT
    formula is satisfied by the input. The exactly-1 version
    requires exactly one literal out of every clause to be satisfied.
    """
    num_clauses = len(exactly_1_3_sat_formula)
    if (num_clauses > 3):
        raise ValueError('We only allow at most 3 clauses')
    for (k, clause) in enumerate(exactly_1_3_sat_formula):
        # This loop ensures aux[k] is 1 if an odd number of literals
        # are true
        for literal in clause:
            if literal > 0:
                circuit.cx(f_in[literal-1], aux[k])
            else:
                circuit.x(f_in[-literal-1])
                circuit.cx(f_in[-literal-1], aux[k])
        # Flip aux[k] if all literals are true, using auxiliary qubit
        # (ancilla) aux[num_clauses]
        circuit.ccx(f_in[0], f_in[1], aux[num_clauses])
        circuit.ccx(f_in[2], aux[num_clauses], aux[k])
        # Flip back to reverse state of negative literals and ancilla
        circuit.ccx(f_in[0], f_in[1], aux[num_clauses])
        for literal in clause:
            if literal < 0:
                circuit.x(f_in[-literal-1])
    # The formula is satisfied if and only if all auxiliary qubits
    # except aux[num_clauses] are 1
    if (num_clauses == 1):
        circuit.cx(aux[0], f_out[0])
    elif (num_clauses == 2):
        circuit.ccx(aux[0], aux[1], f_out[0])
    elif (num_clauses == 3):
        circuit.ccx(aux[0], aux[1], aux[num_clauses])
        circuit.ccx(aux[2], aux[num_clauses], f_out[0])
        circuit.ccx(aux[0], aux[1], aux[num_clauses])
    # Flip back any auxiliary qubits to make sure state is consistent
    # for future executions of this routine; same loop as above.
    for (k, clause) in enumerate(exactly_1_3_sat_formula):
        for literal in clause:
            if literal > 0:
                circuit.cx(f_in[literal-1], aux[k])
            else:
                circuit.x(f_in[-literal-1])
                circuit.cx(f_in[-literal-1], aux[k])
        circuit.ccx(f_in[0], f_in[1], aux[num_clauses])
        circuit.ccx(f_in[2], aux[num_clauses], aux[k])
        circuit.ccx(f_in[0], f_in[1], aux[num_clauses])
        for literal in clause:
            if literal < 0:
                circuit.x(f_in[-literal-1])
```

6.3. Inversion about the Average. The inversion about the average is discussed in sections 5.2.3–5.2.4. This can be implemented as follows:

```
def inversion_about_average(circuit, f_in, n):
    """Apply inversion about the average step of Grover's algorithm."""
    # Hadamards everywhere
    for j in range(n):
        circuit.h(f_in[j])
    # D matrix: flips the sign of the state |00...00> only
    for j in range(n):
        circuit.x(f_in[j])
    n_controlled_Z(circuit, [f_in[j] for j in range(n-1)], f_in[n-1])
    for j in range(n):
        circuit.x(f_in[j])
    # Hadamards everywhere again
    for j in range(n):
        circuit.h(f_in[j])
```

The above routine requires a $\mathrm{C}^{n-1}Z$ gate; we implement it, for three qubits, using a CCNOT gate and two Hadamards. We raise an exception if there are more than two controls, which are not necessary for this example:

```
def n_controlled_Z(circuit, controls, target):
    """Implement a Z gate with multiple controls"""
    if (len(controls) > 2):
        raise ValueError('The controlled Z with more than 2 ' +
                         'controls is not implemented')
    elif (len(controls) == 1):
        circuit.h(target)
        circuit.cx(controls[0], target)
        circuit.h(target)
    elif (len(controls) == 2):
        circuit.h(target)
        circuit.ccx(controls[0], controls[1], target)
        circuit.h(target)
```

6.4. Putting Everything Together. To run Grover's algorithm, we have to initialize a quantum circuit with the desired number of qubits and apply the routines described above. Notice that for three qubits the optimal number of Grover iterations is two; see (5.4) (the $\sin\theta$ approximation is not accurate for $n = 3$, but doing the calculations more carefully, we can verify that two iterations is still optimal). Including the small, necessary setup to initialize a quantum circuit with Qiskit ("classical registers" are used to store the result of a measurement), this results in the following code:

```
import sys
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
from qiskit import compiler, Aer
from qiskit.tools import visualization

# Make a quantum program for the n-bit Grover search.
n = 3
# Exactly-1 3-SAT formula to be satisfied, in conjunctive
# normal form. We represent literals with integers, positive or
# negative to indicate a boolean variable or its negation.
exactly_1_3_sat_formula = [[1, 2, -3], [-1, -2, -3], [-1, 2, 3]]

# Define three quantum registers: 'f_in' is the search space (input
# to the function f), 'f_out' is bit used for the output of function
# f, aux are the auxiliary bits used by f to perform its
# computation.
```

```
f_in = QuantumRegister(n)
f_out = QuantumRegister(1)
aux = QuantumRegister(len(exactly_1_3_sat_formula) + 1)
# One classical register to store the result of a measurement
ans = ClassicalRegister(n)
# Create quantum circuit with the quantum and classical registers
# defined above
qc = QuantumCircuit(f_in, f_out, aux, ans, name='grover')

input_state(qc, f_in, f_out, n)
# Apply two full iterations
black_box_u_f(qc, f_in, f_out, aux, n, exactly_1_3_sat_formula)
inversion_about_average(qc, f_in, n)
black_box_u_f(qc, f_in, f_out, aux, n, exactly_1_3_sat_formula)
inversion_about_average(qc, f_in, n)
# Measure the output register in the computational basis
for j in range(n):
    qc.measure(f_in[j], ans[j])

# Create an instance of the local quantum simulator
quantum_simulator = Aer.get_backend('qasm_simulator')
# Compile the circuit into "quantum object code" that can be
# executed on the simulator
qobj = compiler.assemble(qc, quantum_simulator, shots=2048)
# Execute and store the results. Note that this could take some
# time (up to a few minutes, depending on the machine)
job = quantum_simulator.run(qobj)
result = job.result()
# Get counts
counts = result.get_counts('grover')
print('Observed measurement outcomes:')
print('string | count')
for key in sorted(counts):
    print(' {:>5s}    {:d}'.format(key, counts[key]))

# Plot histogram
figure = visualization.plot_histogram(counts)
print()
# We can display the histogram with figure.show() if matplotlib is
# properly configured. Instead, we write it to file.
figure.savefig('groverhist.png')
print('Histogram saved as groverhist.png')
```

That's it! We have successfully executed Grover's algorithm. The resulting histogram is given in Figure 23: with only two calls to the black-box function U_f, we sample the correct string 101 (the only solution of (6.1)) with probability $\approx 95\%$. The argument `shots` given to the `assemble()` function determines the number of samples extracted from the quantum state, i.e., the quantum circuit is executed that many times, each time performing a measurement (and therefore potentially obtaining different outcomes).

The Qiskit allows the running of experiments on real quantum computing hardware accessible on the cloud via the IBM Q experience, changing the backend object used to run the experiment. In this example, we used a classical simulation of the quantum computer executed locally (via the `qasm_simulator` backend); such a simulation is only able to scale up to a dozen qubits or so (a regular laptop should be able to simulate $\approx$ 18 qubits, but the computation can be slow after 12–14 qubits).

To run experiments on a real device, one first needs an account on the IBM Q

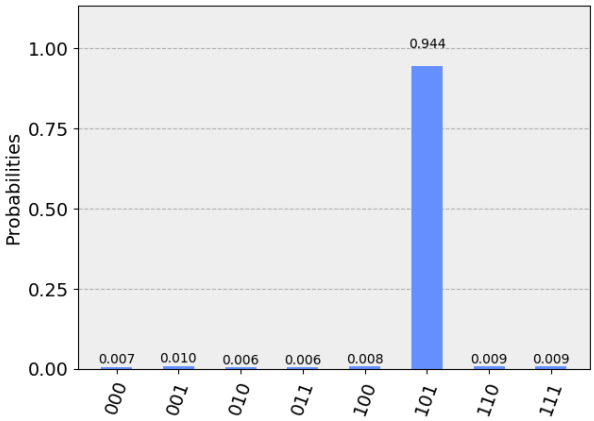


Fig. 23 *Histogram of output probabilities.*

experience, after which `IBMQ` can be loaded from Qiskit and one can access the list of available remote backends—which includes some of IBM's devices. More detailed instructions are available on Qiskit's webpage. We should also remark that the current generation of quantum computing devices has limited qubit connectivity, i.e., it only allows the application of 2-qubit gates between certain pairs of qubits. The code listed in this paper assumes an all-to-all connectivity, which makes for much simpler code. The interested reader can modify the code to allow for limited connectivity, using SWAP gates when necessary.

Another extension of the code that we leave as an exercise (and is perhaps simpler than the above exercise) is to allow an arbitrary number of Boolean variables and clauses in the Exactly-1 3-SAT formula. For this extension, we need a controlled NOT with an arbitrary number of controls. Such a gate will be used in two places: first, to implement the function U_f (in our approach, we use one auxiliary qubit per clause and a final controlled NOT operation to check that all clauses are $|1\rangle$); second, to implement the n-controlled Z. One way to implement a controlled NOT with an arbitrary number of controls is to apply the same idea as the circuit in Figure 22 for the triply controlled NOT. This will require extra qubits, which will slow down the classical simulation of the resulting circuit.

7. Further Reading. In this paper we have used several notational devices to help the reader, but they are usually not employed in the quantum computing literature. We list them here.

- The subscript for bra-ket vectors to indicate the dimension of the space, e.g., $|\psi\rangle_q$ for 2^q-dimensional vectors. Typically, the dimension of the space is defined elsewhere and can be understood from the context. Whenever subscripts for kets are used, it is normally to address registers.
- The vector arrow, e.g., $\vec{j}$, to indicate binary strings. Typically, binary strings are not distinguished from other mathematical symbols and they can be identified from the context.
- The use of Roman letters for basis vectors and Greek letters for general, i.e., possibly not basis, vectors.

- The notation for the probability of observing the measurement outcomes: this paper makes explicit the qubit(s) that is (are) being measured, but typically this is only defined by the context.

Finally, the all-zero binary string of dimension q is normally denoted 0^q, rather than $\vec{0}$.

We end this paper with some pointers to papers describing quantum algorithms that provide a speedup with respect to classical computation, in various areas of computing. This list is by no means exhaustive: it merely serves the purpose of giving the reader some ideas about existing work showcasing the power of quantum computation. Additional references can be found from the papers listed below, or on the arXiv, where most of the quantum literature is available.

The first article that we mention is of course Shor's paper on integer factorization [35]. Shor's algorithm determines the prime factors of an integer using a combination of a classical algorithm and a quantum subroutine for the following period finding problem: given integers a, n with $a < n$, find the smallest integer r such that $a^x \mod n = a^{x+r} \mod n$ for all x. We call this "period finding" because it can be seen as determining the period of the function $f(x) = a^x \mod n$.

For readers who are familiar with topology, the work by Aharonov, Jones, and Landau on the approximation of Jones polynomial [2] will be of great interest. The Jones polynomial is an invariant of an oriented knot. It is known that evaluating the Jones polynomial exactly is very difficult: it is a #P-hard problem, and it is not expected to admit a polynomial-time classical algorithm. The paper [2] shows how to approximately solve the problem in polynomial time using a quantum computer—an easier task because of the approximation, but one for which no classical algorithm is known. Furthermore, computing such an approximation is a BQP-complete problem, i.e., it solves a problem that is as hard as any other problem that admits an efficient quantum algorithm. It is believed that BQP-complete problems cannot be solved efficiently on classical computers, because that would imply BQP = P, which in turn implies that there exists a polynomial-time classical algorithm for integer factorization. Thus, surprisingly, the computational power of quantum computing can be rephrased in terms of the ability to approximate this topological quantity.

Another BQP-complete problem that has received a lot of attention is Hamiltonian simulation. This is the problem of simulating the time evolution of a quantum system, for which quantum computers were originally proposed [19]: it has been known since the early days that a quantum computer can efficiently solve it [27], whereas all known classical algorithms require exponential time. There are many papers on this topic; we refer to the very recent [22] for an entry point with a good list of references.

Exponential speedups can also be obtained for easier-to-describe problems. As an example, we mention the work of Harrow, Hassidim, and Lloyd [23] on the solution of linear systems in logarithmic time (although attaining logarithmic time requires several assumptions on the input and the output of the algorithm, because parsing the equations in the linear system already trivially requires linear time), and several papers discussing quantum acceleration for random walks on graph [13], which find many applications in computer science, e.g., [5].

Acknowledgments. We are extremely grateful to two anonymous referees, whose patience and numerous detailed remarks on an earlier draft of this manuscript helped significantly improve its quality, and to Sergey Bravyi for many illuminating discussions.

REFERENCES

[1] D. Aharonov, M. Ben-Or, E. Eban, and U. Mahadev, *Interactive Proofs for Quantum Computations*, preprint, https://arxiv.org/abs/1704.04487, 2017. (Cited on p. 965)

[2] D. Aharonov, V. Jones, and Z. Landau, *A polynomial quantum algorithm for approximating the Jones polynomial*, Algorithmica, 55 (2009), pp. 395–421. (Cited on p. 979)

[3] D. Aharonov, W. van Dam, J. Kempe, Z. Landau, S. Lloyd, and O. Regev, *Adiabatic quantum computation is equivalent to standard quantum computation*, SIAM Rev., 50 (2008), pp. 755–787, https://doi.org/10.1137/080734479. (Cited on p. 939)

[4] G. Aleksandrowicz, T. Alexander, P. Barkoutsos, L. Bello, Y. Ben-Haim, D. Bucher, F. J. Cabrera-Hernández, J. Carballo-Franquis, A. Chen, C.-F. Chen, J. M. Chow, A. D. Córcoles-Gonzales, A. J. Cross, A. Cross, J. Cruz-Benito, C. Culver, S. D. L. P. González, E. D. L. Torre, D. Ding, E. Dumitrescu, I. Duran, P. Eendebak, M. Everitt, I. F. Sertage, A. Frisch, A. Fuhrer, J. Gambetta, B. G. Gago, J. Gomez-Mosquera, D. Greenberg, I. Hamamura, V. Havlicek, J. Hellmers, Ł. Herok, H. Horii, S. Hu, T. Imamichi, T. Itoko, A. Javadi-Abhari, N. Kanazawa, A. Karazeev, K. Krsulich, P. Liu, Y. Luh, Y. Maeng, M. Marques, F. J. Martín-Fernández, D. T. McClure, D. McKay, S. Meesala, A. Mezzacapo, N. Moll, D. M. Rodríguez, G. Nannicini, P. Nation, P. Ollitrault, L. J. O'Riordan, H. Paik, J. Pérez, A. Phan, M. Pistoia, V. Prutyanov, M. Reuter, J. Rice, A. R. Davila, R. H. P. Rudy, M. Ryu, N. Sathaye, C. Schnabel, E. Schoute, K. Setia, Y. Shi, A. Silva, Y. Siraichi, S. Sivarajah, J. A. Smolin, M. Soeken, H. Takahashi, I. Tavernelli, C. Taylor, P. Taylour, K. Trabing, M. Treinish, W. Turner, D. Vogt-Lee, C. Vuillot, J. A. Wildstrom, J. Wilson, E. Winston, C. Wood, S. Wood, S. Wörner, I. Y. Akhalwaya, and C. Zoufal, *Qiskit: An Open-Source Framework for Quantum Computing*, 2019, https://doi.org/10.5281/zenodo.2562110. (Cited on p. 972)

[5] A. Ambainis, *Quantum walk algorithm for element distinctness*, SIAM J. Comput., 37 (2007), pp. 210–239, https://doi.org/10.1137/S0097539705447311. (Cited on p. 979)

[6] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, and H. Weinfurter, *Elementary gates for quantum computation*, Phys. Rev. A, 52 (1995), pp. 3457–3467. (Cited on pp. 969, 970)

[7] C. H. Bennett, *Logical reversibility of computation*, IBM J. Res. Dev., 17 (1973), pp. 525–532. (Cited on p. 947)

[8] C. H. Bennett, E. Bernstein, G. Brassard, and U. Vazirani, *Strengths and weaknesses of quantum computing*, SIAM J. Comput., 26 (1997), pp. 1510–1523, https://doi.org/10.1137/S0097539796300933. (Cited on p. 960)

[9] E. Bernstein and U. Vazirani, *Quantum complexity theory*, SIAM J. Comput., 26 (1997), pp. 1411–1473, https://doi.org/10.1137/S0097539796300921. (Cited on pp. 938, 961)

[10] G. Brassard, P. Hoyer, M. Mosca, and A. Tapp, *Quantum amplitude amplification and estimation*, Contemp. Math., 305 (2002), pp. 53–74. (Cited on p. 965)

[11] A. Broadbent, J. Fitzsimons, and E. Kashefi, *Universal blind quantum computation*, in 50th Annual IEEE Symposium on Foundations of Computer Science, FOCS'09, IEEE Press, 2009, pp. 517–526. (Cited on p. 965)

[12] D. Castelvecchi, *Quantum computers ready to leap out of the lab in* 2017, Nature News, 541 (2017), p. 9. (Cited on pp. 938, 958)

[13] A. M. Childs, R. Cleve, E. Deotto, E. Farhi, S. Gutmann, and D. A. Spielman, *Exponential algorithmic speedup by a quantum walk*, in Proceedings of the 35th Annual ACM Symposium on Theory of Computing, ACM, 2003, pp. 59–68. (Cited on p. 979)

[14] V. Coffman, J. Kundu, and W. K. Wootters, *Distributed entanglement*, Phys. Rev. A, 61 (2000), art. 052306. (Cited on p. 946)

[15] C. M. Dawson and M. A. Nielsen, *The Solovay-Kitaev algorithm*, Quantum Inform. Comput., 6 (2006), pp. 81–95. (Cited on p. 958)

[16] D. Deutsch, *Quantum theory, the Church-Turing principle and the universal quantum computer*, Proc. Roy. Soc. London Ser. A, 400 (1985), pp. 97–117. (Cited on pp. 938, 947)

[17] D. Deutsch and R. Jozsa, *Rapid solution of problems by quantum computation*, Proc. Roy. Soc. London Ser. A, 439 (1992), pp. 553–558. (Cited on p. 961)

[18] M. H. Devoret and R. J. Schoelkopf, *Superconducting circuits for quantum information: An outlook*, Science, 339 (2013), pp. 1169–1174. (Cited on p. 958)

[19] R. P. Feynman, *Simulating physics with computers*, Internat. J. Theoret. Phys., 21 (1982), pp. 467–488. (Cited on p. 979)

[20] M. R. Garey and D. S. Johnson, *Computers and Intractability*, W. H. Freeman, New York, 1972. (Cited on p. 973)

[21] L. K. Grover, *A fast quantum mechanical algorithm for database search*, in Proceedings of the 28th Annual ACM Symposium on Theory of Computing, ACM, 1996, pp. 212–219. (Cited on pp. 965, 966)
[22] J. Haah, M. B. Hastings, R. Kothari, and G. H. Low, *Quantum algorithm for simulating real time evolution of lattice Hamiltonians*, in 59th Annual IEEE Symposium on Foundations of Computer Science (FOCS), IEEE, 2018, pp. 350–360. (Cited on p. 979)
[23] A. W. Harrow, A. Hassidim, and S. Lloyd, *Quantum algorithm for linear systems of equations*, Phys. Rev. Lett., 103 (2009), art. 150502, https://doi.org/10.1103/PhysRevLett.103.150502. (Cited on p. 979)
[24] A. S. Holevo, *Bounds for the quantity of information transmitted by a quantum communication channel*, Probl. Inform. Transmission, 9 (1973), pp. 177–183. (Cited on p. 943)
[25] A. Y. Kitaev, *Quantum computations: Algorithms and error correction*, Russian Math. Surveys, 52 (1997), pp. 1191–1249. (Cited on p. 958)
[26] V. Kliuchnikov, D. Maslov, and M. Mosca, *Practical approximation of single-qubit unitaries by single-qubit quantum Clifford and T circuits*, IEEE Trans. Comput., 65 (2016), pp. 161–172. (Cited on p. 958)
[27] S. Lloyd, *Universal quantum simulators*, Science, 273 (1996), pp. 1073–1078. (Cited on p. 979)
[28] U. Mahadev, *Classical verification of quantum computations*, in 59th Annual IEEE Symposium on Foundations of Computer Science, FOCS'18, IEEE, 2018, pp. 259–267. (Cited on p. 965)
[29] N. D. Mermin, *Quantum Computer Science: An Introduction*, Cambridge University Press, 2007. (Cited on pp. 937, 964)
[30] M. A. Nielsen and I. Chuang, *Quantum Computation and Quantum Information*, Cambridge University Press, 2002. (Cited on pp. 937, 958, 959)
[31] B. W. Reichardt, F. Unger, and U. Vazirani, *Classical command of quantum systems*, Nature, 496 (2013), p. 456. (Cited on p. 965)
[32] E. Rieffel and W. Polak, *An introduction to quantum computing for non-physicists*, ACM Comput. Surveys, 32 (2000), pp. 300–335. (Cited on p. 937)
[33] E. G. Rieffel and W. H. Polak, *Quantum Computing: A Gentle Introduction*, MIT Press, 2011. (Cited on p. 937)
[34] P. Selinger, *Efficient Clifford+T approximation of single-qubit operators*, Quantum Inform. Comput., 15 (2015), pp. 159–180. (Cited on p. 958)
[35] P. W. Shor, *Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer*, SIAM J. Comput., 26 (1997), pp. 1484–1509, https://doi.org/10.1137/S0097539795293172. (Cited on pp. 961, 979)
[36] D. R. Simon, *On the power of quantum computation*, SIAM J. Comput., 26 (1997), pp. 1474–1483, https://doi.org/10.1137/S0097539796298637. (Cited on pp. 961, 962)
[37] A. C.-C. Yao, *Quantum circuit complexity*, in 34th Annual IEEE Symposium on Foundations of Computer Science, FOCS'93, IEEE, 1993, pp. 352–361. (Cited on p. 938)