Introduction to quantum computation

Jérémie Roland

March 17, 2015

1 Classical computation

1.1 A brief history of theoretical computer science

Long before the first computers were built, at the dawn of mathematics, men realized that many everyday life problems could be translated into mathematical problems, and that some would be more complex to solve than others. For instance, a farmer could easily evaluate the area of his triangular field, but it was much more difficult to find the optimal way to use it, knowing that he could only cultivate particular vegetable patches in particular seasons of the year, that different seeds had different costs but could also lead to different profits, and many other parameters that should be taken into consideration. Intuitively, we understand that the first problem would take less time to solve than the latter, and therefore they should be classified in different classes of complexity.

However, one had to wait until the twentieth century to clarify this matter. Actually, the first step was taken in the middle of the nineteenth century by Charles Babbage (1791-1871), who conceived his Analytical Engine, an apparatus that would be able to perform any mathematical calculation. Nonetheless, technology lacked at that time and even when his son continued his work after he died, his theoretical model of the Analytical Engine could never been practically built and the prototypes could only solve a few problems with very frequent errors.

Another important step was performed by Kurt Gödel (1906-1978) when he gave a—rather surprising—answer to a question asked by David Hilbert (1862-1943), who realized that before trying to prove that a mathematical proposition was true or not, one should ask if such a proof was actually always achievable (this question led to Hilbert's 2nd problem, one of 23 mathematical problems proposed by Hilbert in 1900 [Hil02]). He thus questioned the intuition, widely spread at the time among mathematicians, that mathematics were complete, in the sense that within a mathematical theory, any proposition could be proved to be either true or false. Surprisingly, Gödel showed that this was not the case by establishing the existence of mathematical propositions that were undecidable, meaning that they could be neither proved nor disproved (a statement which is now known as Gödel's incompleteness theorem [Göd31]). Equivalently, this refuted the idea that any properly defined mathematical function was computable or that, for every properly defined mathematical problem, it was either possible to find a solution, or to prove that none exists.

Relying on Gödel's work, it is precisely the problem of establishing the decidability of mathematical propositions, instead of their truth, that Alan Turing (1912-1954) addressed when he proposed his model that is now widely known as the Universal Turing Machine [Tur36]. Building up on ideas developed by Babbage to devise his Analytical Engine, he defined a theoretical model of machine that was sufficiently complicated to address any mathematical problem but sufficiently simple to be analytically studied (a schematic representation of a Turing Machine is given in Fig. 1). Even though at that time a "computer" looked actually more like a mathematician writing equations on a piece of paper, his model proved to be sufficiently general to be "computationally" equivalent to other models, such as the circuit-model we will consider later, or even to nowadays computers. In particular, considering the decidability problem, it is now widely believed that all these models allow the same functions to be computed, what Alan Turing and

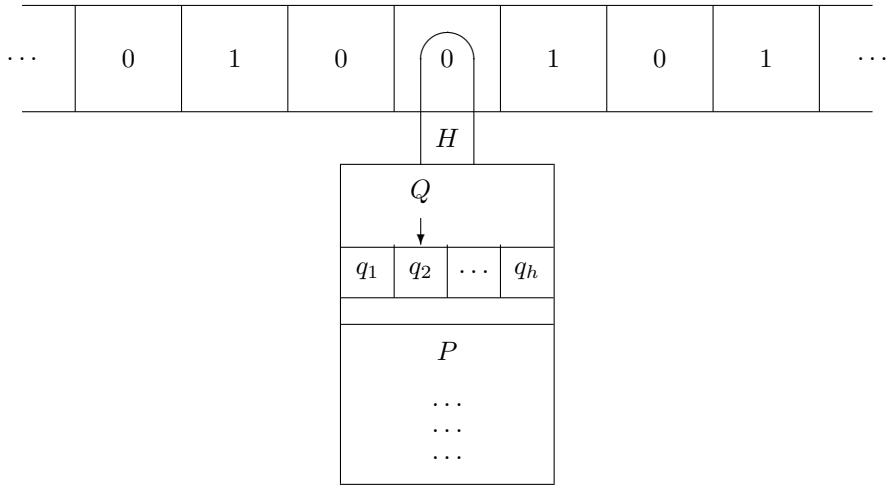

Figure 1: Schematic representation of a Turing Machine: The machine acts following a fixed program P, and an internal state Q that may vary during the computation, taking values in a finite set $\{q_1, q_2, \ldots, q_h\}$. Thanks to a read-write tape-head H, it may interact with a tape, used like a computer memory, printed with symbols s_i. The computation consists in successive identical steps. At each step, the machine reads the symbol s_i written on the tape at the current position i of the head. Then, depending on s_i, the program P and the current internal state q, it overwrites the symbol with a new value s'_i, changes its internal state to q' then moves the head H left or right. At some point, if the machine enters the special state q_h, it halts. The computation is then finished and the output may be read on the tape.

Alonzo Church summarized, independently and both in 1936, in a thesis that took their name [Chu36]:

> Every function "which would *naturally* be regarded as computable" can be computed by the Universal Turing Machine.

Actually, the *Church-Turing Thesis* is not a theorem, but rather an assertion about the Real World, as the set of functions being *naturally* regarded as computable is not defined rigorously. Intuitively, it corresponds to the set of functions that may be computed with any realistic physical means. Consequently, this thesis remains unproven, but the fact that no counter-example could be found in spite of numerous attempts convinces most theoreticians of its validity.

Following this remarkable result, we may now classify problems into two groups, whether they are solvable (if they reduce to a decidable mathematical proposition or a computable function) or not, but we have not yet defined how, among solvable problems, discriminate between easy or complex ones. An answer to this question was given by Alan Cobham and Jack Edmonds [Cob65, Edm65], who defined the complexity of an algorithm as the number of elementary operations it requires to be run on a Universal Turing Machine. A problem which requires a number of operations that grows exponentially with the size of the input would then be considered as complex, or *intractable* while a polynomial growth would correspond to *tractable* problems. Of course, this supposes that we know the optimal algorithm to solve a problem (for instance, there exist problems for which only exponential algorithms are known, but the existence of a polynomial algorithm is not proved to be impossible). Remarkably, it seemed that the translation of any algorithm initially formulated within another computational model into the Universal Turing Machine model could only increase the number of basics steps polynomially, so that this definition of complexity would be model-independent. However, while Church-Turing thesis survived against many attempts to find a counter-example, it was not the case for this conjecture.

The first alarm was given when it was shown that an analog computation model could outperform the Universal Turing Machine for some tasks. Nonetheless, this model required the use of

continuous variables with arbitrary large precision, which would not be practically feasible, and it was shown that whenever errors would occur during the computation, the analog computer could not beat the Universal Turing Machine anymore. While the analog computation model did therefore not contradict the universality of the definition of complexity based on the Universal Turing Machine when considering realistic computations, a new crisis occurred when Robert Solovay and Volker Strassen showed that it was possible to test the primality of a given integer with a *randomized* algorithm [SS77]. Their algorithm used randomness at some steps of the computation and could tell in polynomial time either that the integer was composite with certainty or prime with some large probability. Repeating the algorithm a few times, it was therefore possible to find wether an integer was prime or not with near certainty, while, at that time, no deterministic polynomial algorithm to solve this problem was known (a deterministic polynomial algorithm for this problem has been discovered in 2004 [AKS04]). This suggested that randomized algorithms could beat deterministic ones, which inspired many other successful results. To keep an accurate definition of computation complexity, the Universal Turing Machine was generalized by allowing the use of randomness along the computation, which led to the *Strong Church-Turing Thesis*:

> Any model of computation can be simulated on a *probabilistic* Turing Machine with at most a polynomial increase in the number of elementary operations required.

The fact that the discovery of a new type of algorithms required a modification of the Strong Church-Turing Thesis suggests that we must stay cautious. Would it be possible that there exists an even more general type of Turing Machine? To answer this question, one approach would be to look for the basic principles that determine what is a possible computation... Realizing that any type of "computer", would it be a mathematician scribbling on a piece of paper, Babbage's Analytical Engine or a state-of-the art personal computer, is actually a physical system, we understand that possible computations are ultimately limited by the laws of Nature. However, while we know from the revolution that physics lived during the last century that Nature is quantum mechanical, the concepts of quantum mechanics are rather counter-intuitive to the human mind, which is much more used to the *classical* (as opposed to *quantum*) properties of large physical systems. Accordingly, it is within a *classical* view of Nature that Turing developed his ideas. Therefore, it is possible that a Quantum Turing Machine could be more powerful than its classical equivalent. Since the dynamics of a Quantum Turing Machine would be governed, as any other quantum system, by the Schrödinger equation, it is clear that it could be simulated on a classical Turing Machine just by numerically integrating this equation, so that the introduction of this Quantum Machine would not modify the set of computable functions (that is, the Weak Church-Turing Thesis). Nonetheless, this simulation would involve the evaluation of a number of complex amplitudes that increases exponentially with the size of the input, so that it would not be efficient. It is therefore possible that there exist polynomial quantum algorithms for problems that are thought to be classically intractable. The quest for such algorithms is the main goal of the flourishing field of quantum computation.

In this course, we will first briefly summarize some notions of classical computation that will be necessary for our purposes, such as the ideas of algorithm, universal gates and circuit complexity. Then, after having reviewed the postulates of quantum mechanics, we will generalize these notions to quantum computation.

1.2 Circuit model

The goal of computation is to design efficient analytical methods, called *algorithms*, to solve mathematical problems. Before considering algorithms, let us first focus on the problems. A general problem is given as a mathematical description of what is sought (the solution or *output*) depending on some unspecified variable, or *input*. The specification of this variable fixes the particular *instance* of the problem. For example, the factorization problem may be defined in the following way: "Given an integer x, find one of its factor y". Different values of the input x will correspond to different instances of this general problem. The value of x could be written with different symbols and take the form of a number written in decimal notation or a word made of

letters from the Latin alphabet. To fix a standard, we will use the binary notation, generated by the minimal set of symbols $\{0,1\}$. Each binary variable is then called a *bit*, for "binary digit", and a sequence of n bits allows to define 2^n different instances. We will often refer to n as the *size* of the problem (it is actually the size of the input written in binary notation).

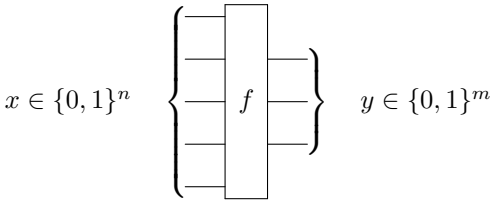

Figure 2: Algorithm with input x and output y.

An algorithm is an analytical method that, given the input x of a problem, provides an output y that satisfies the definition of the solution of the problem. Since the output y may be, exactly as the input x, written in binary notation, an algorithm corresponds to the evaluation of a function $f : \{0,1\}^n \to \{0,1\}^m : x \mapsto y = f(x)$. If this function is computable we know that the problem may be solved using a Universal Turing Machine. However, from now on we will consider another model of computation that may be proved to be equivalent, namely the *circuit model*.

Writing the m bit output $y = (y_1, \ldots, y_m)$, the function f defines m functions with one bit output $f_k : \{0,1\}^n \to \{0,1\} : x \mapsto y_k = f_k(x)$ $(k = 1, \ldots, m)$. Each of these functions may then be rewritten as a boolean expression of the input bits $x = (x_1, \ldots, x_n)$ combined with the boolean operators NOT ($\neg$), AND ($\wedge$) and OR ($\vee$). These boolean expressions may be evaluated using a circuit made of wires carrying the value of bits between logical gates corresponding to the boolean operators (see Fig. 3).

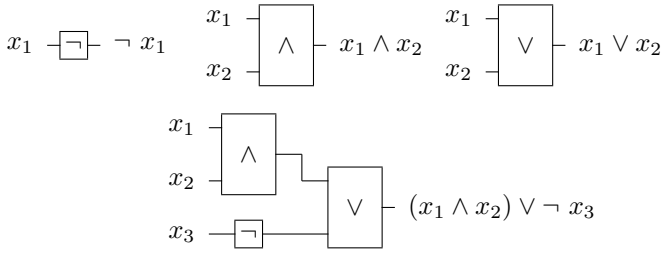

Figure 3: The classical gates NOT ($\neg$), AND ($\wedge$), and OR ($\vee$) and a small circuit implementing the boolean expression $(x_1 \wedge x_2) \vee \neg\, x_3$.

While each of the boolean operators cited above corresponds to a particular logical gate, we may define any logical gate as a function $g : \{0,1\}^n \to \{0,1\}^m : x \mapsto g(x)$. In particular, apart from the NOT, AND and OR gates, the practical implementation of the circuits have implicitely assumed the existence of two other gates: FANOUT, which duplicates a bit and is necessary if a bit appears more than once in the boolean expression, and SWAP, which swaps two wires and is required to be able to apply a logical gate on two spatially separated bits (see Fig. 4).

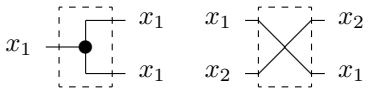

Figure 4: FANOUT and SWAP gates, which respectively duplicates a bit and swaps two bits.

1.3 Universal set of gates

One important issue is to find a *universal* set of gates, meaning that, combined together, these gates are able to simulate any other logical gate. This is the case of the set {NOT, AND, OR, FANOUT, SWAP} that we just described, but there exists other equivalent universal sets. One useful trick to simulate a logical gate from others is to make use of an extra bit with a known value, called an *ancilla*. For instance, as shown in Fig. 5, it is possible to simulate the logical gate AND with one ancilla bit with value 1 and two uses of the NAND gate ("NOT AND", which outputs 0 if and only if both input bits are 1). Actually, it is possible to show that one can simulate any logical gate using ancilla bits and the set {NAND, FANOUT, SWAP}, which is therefore universal.

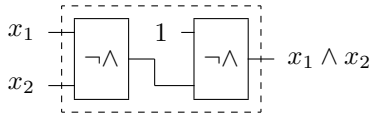

Figure 5: Simulation of an AND gate ($\wedge$) from two NAND gates ($\neg\wedge$) and an ancilla with value 0.

1.4 Circuit complexity

Let us now turn to the notion of complexity, which, following the Strong Church-Turing Thesis, is related to the number of elementary operations needed to solve the problem with a Turing Machine. Let us translate this idea into the picture of the circuit model of computation.

Within this model, an algorithm corresponds to a family of circuits C_n, made of gates from a universal set, for all sizes of the input n. To derive a proper notion of complexity, we add a *uniformity* condition, which requires that the design of the circuit C_n may be computed efficiently (in polynomial time versus the size n) for instance on a Turing Machine. This prevents to hide the hardness of a problem within a complex procedure to design the circuit and even more the possibility to compute uncomputable functions. For such a uniform family of circuits, the complexity will be defined as the *size* of the circuit C_n, that is, the number of elementary gates in the circuit. Following this idea, we consider that the size of the circuit corresponds to the time it takes to run the algorithm.

Actually, we are not interested in the exact size of the quantum circuit, which for instance may vary for different universal sets of gates, but mostly on the way it scales versus the size of the input n. Following the Strong Church-Turing Thesis, an algorithm is called *efficient* if its complexity grows polynomially with n. To precise this notion of scaling, we will use the asymptotic notation big-O:

$$f(n) = O(g(n)) \Leftrightarrow \exists c \geq 0, n_0 \geq 0 \text{ s.t. } 0 \leq f(n) \leq cg(n) \quad \forall n \geq n_0. \tag{1}$$

Intuitively, it means that $f(n)$ grows slower (or equally) with n than $g(n)$. We will also use the inverse notation big-Ω:

$$f(n) = \Omega(g(n)) \Leftrightarrow \exists c \geq 0, n_0 \geq 0 \text{ s.t. } 0 \leq cg(n) \leq f(n) \quad \forall n \geq n_0. \tag{2}$$

that means that $f(n)$ grows faster (or equally) with n than $g(n)$.

Practically, a tractable problem admits efficient algorithms, that show a complexity scaling as $O(n^c)$ for some exponent c, while an intractable problem may not be solved by any algorithm with a complexity that scales as $\Omega(n^c)$, no matter how large c is. More generally, the big-O notation is often used to qualify the worst-case behavior of an algorithm, while the big-Ω allows to specify lower bounds on the complexity of all algorithms able to solve a particular problem. The ideal case occurs when the worst-case behavior of the best known algorithm saturates the lower bound, in which case we know that this scaling is really optimal and we may use the big-Θ notation

$$f(n) = \Theta(g(n)) \Leftrightarrow f(n) = O(g(n)) \text{ and } f(n) = \Omega(g(n)), \tag{3}$$

meaning that both functions scale similarly with n.

Let us note that the following properties immediately follow from the definitions

$$f(n) = O(g(n)) \quad \Leftrightarrow \quad g(n) = \Omega(f(n)) \tag{4}$$
$$f(n) = \Theta(g(n)) \quad \Leftrightarrow \quad g(n) = \Theta(f(n)). \tag{5}$$

Moreover, while these definitions apply to the limit of large n, that is, $n \to \infty$, we may also study a scaling in the limit of, for instance, a small error probability ε, extending the definition of the big-O notation to the case $\varepsilon \to 0$:

$$f(\varepsilon) = O(g(\varepsilon)) \Leftrightarrow \exists c \geq 0, \varepsilon_0 \geq 0 \text{ s.t. } 0 \leq f(\varepsilon) \leq cg(\varepsilon) \quad \forall \varepsilon \leq \varepsilon_0. \tag{6}$$

1.5 Complexity classes

Now that we have defined the notions of logical circuit and complexity, we may classify mathematical problems according to the complexity of the optimal circuit that allows to solve them. This is the goal of complexity theory, a broad field of information science of which we only sketch here the basic ideas that will be helpful for the sake of this course. Complexity theory classifies problems into different *complexity classes*, of which we will only see the few that will be relevant to us. Note that many complexity classes are defined for so-called *decision problems*, that is problems with a yes/no answer, or equivalently, functions with a binary output (setting yes=1 and no=0).

On one hand of the complexity scale, are easy problems that may be solved in polynomial time and define the complexity class P, for *polynomial*. Unfortunately, there are of course numerous problems for which only exponential algorithms are known. Among these problems, an important sub-class is $PSPACE$ which brings together those admitting a possibly exponentially deep circuit that works on a polynomially large register of bits, that is, they may require exponential time but only *polynomial space*.

While it is still unproven whether $P \neq PSPACE$, it is one of the most important conjecture of complexity theory. Whatsoever, complexity classes that would be intermediate between P and $PSPACE$ may be theoretically defined, such as the class of *non-deterministic polynomial* problems, or NP, which includes problems that may possibly only be solved in exponential time, but, roughly speaking, whose solution may be checked in polynomial time if some data, called a *witness* is provided. More precisely, a problem f is in NP if there exists a polynomial time circuit g such that

$$f(x) = 1 \Leftrightarrow g(x, y) = 1 \text{ for some } y$$

One example would be the FACTORING problem, or more precisely a decision version asking if an integer x admits a prime factor less than some other integer C. Indeed, it may not be possible to factor a large number in polynomial time, but if a factor of the number is given as a witness, this may easily be checked just by performing a division, which takes a polynomial time.

Another example would be the problem of the satisfiability of boolean formulas (SAT), which amounts to determine if a given boolean formula involving a set of n binary variables may be satisfied for some assignments of the variables. In this case, it is possible to check in polynomial time whether the formula is satisfiable if a satisfying assignment is given as a witness. The particularity of this problem is that it has been proved that any other problem in NP may be reduced to it with only polynomial overhead, meaning that this problem is among the most difficult problems in NP. Other problems share this property, they form the class of NP-complete problems, NP-c.

From the definition of NP-completeness, it follows that if a polynomial algorithm was found for any NP-complete problem, all other problems in NP could be solved in polynomial time and the class NP would reduce to the class P. Despite years of attempts, no polynomial algorithm for an NP-complete problem has been found so that it is now widely conjectured that none exists,

although as no proof ruling out the possibility of such an algorithm was derived, the question $P = NP$ remains open.

If we assume that $P \neq NP$, it follows that there exist problems in NP that are neither P (easy) or NP-complete (the hardest NP problems). These problems are therefore of intermediate complexity, and constitute the class NPI. Another important class is co-NP, which is somewhat dual to NP, in the sense that while one can answer a decision problem in NP by exhibiting an example, one can answer a decision problem in co-NP by exhibiting a counter-example. Formally, for a decision problem f in co-NP, there should exists a polynomial circuit g such that

$$f(x) = 1 \Leftrightarrow g(x, y) = 1 \text{ for all } y.$$

For example the SAT problem, which belongs to NP, is equivalent to the question "Is there a satisfying assignment?", which can be answered by providing an example of satisfying assignment. On the other hand, by rephrasing the question in the negative, i.e., "Is there no satsifying assignment?", one obtains a decision problem in co-NP, which can now be answered by providing a counter-example.

Note that if a decision problem is in both NP and co-NP, one can efficiently verify both *yes* and *no* answers. It is conjectured (though not proved) that $NP \neq$co-NP (for example, we can show that a boolean formula is satisfiable by providing a satisfying assignment, but we have no idea how to efficiently prove that it is not satisfiable!). Assuming that $NP \neq$co-NP, it can be shown that no co-NP problems are also NP-complete. This means that problems that are both in P and co-NP, but not believed to be in P, are good candidates for NPI problems. One such problem is the FACTORING problem (or rather its decision version). In the context of quantum computation, problems in NPI would be good candidates for problems being hard for classical computers, but easy for quantum computers.

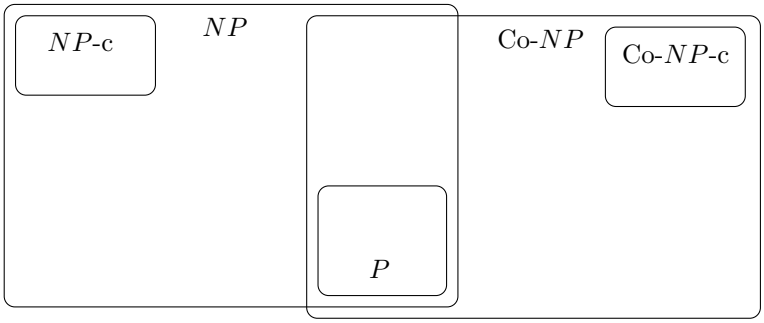

Figure 6: Main complexity classes, assuming $P \neq NP$ and $NP \neq$co-NP.

Another interesting open question is whether it may help to use a randomized algorithm. Such an algorithm would translate into a logical circuit which divides in different paths at some points. One would then have to flip a coin to decide which path to follow. Such a randomized circuit thus relies on a different model of computation, equivalent to the *probabilistic* Universal Turing Machine. This generalized model allows to define new classes of complexity, the most important being the class of *bounded error probabilistic polynomial* problems, or BPP. This class includes problems that may be solved by a probabilistic algorithm in polynomial time with success probability of at least 3/4. Thanks to the Chernoff bound, which shows that, whenever the error probability is smaller than 1/2, it may be reduced exponentially fast just by repeating the algorithm a polynomial number of times, this lower bound of 3/4 is actually arbitrary. Therefore, the class BPP may really be considered as the class of problems that may be solved efficiently, that is, in polynomial time.

1.6 Reversible computation

Even though quantum mechanics involves amplitudes instead of probabilities, quantum processes are inherently probabilistic, so that the introduction of randomness in the computational model took us one step closer to the idea of quantum computation. However, another peculiarity of a quantum evolution is its reversibility.

Considering the circuit model of computation, we see that not all gates are reversible. For instance, while the NOT gate is reversible as one may recover the input bit just by negating the output bit, this is not the case for the XOR gate, which from an input of two bits outputs the single bit corresponding to their sum modulo 2. During the application of this gate, one bit of information is actually lost or, more precisely, erased, which is the cause of irreversibility. On thermodynamical grounds, it may be shown that the erasure of information leads to a consumption of energy, as stated by *Landauer's principle* [Lan61]:

> When a computer erases a single bit of information, the amount of energy dissipated into the environment is at least $k_B T \ln 2$, where k_B is Boltzmann's constant and T is the temperature of the environment.

Even though current computers dissipate much more energy than the lower bound predicted by Landauer's principle, it turns out that this bound is the only ultimate physical limit on the reduction of power consumption, so that, theoretically, it should be possible to design a computer that consumes arbitrarily low power, as long as it behaves reversibly. The question is: is it possible to make any computation reversible? The answer is yes, it suffices to add to each irreversible gate some additional output bits containing the erased information [Ben73]. For instance, the XOR gate may be made reversible by adding a second output bit that takes the value of, say, the first input bit (see Fig. 7). Let us note that the reversible version of XOR then takes a particular form, as it may be viewed as a gate that performs a NOT on one bit if the other bit is set to 1, and leaves it alone otherwise. We call such a gate that performs some operation R on a *target bit* if and only if a *control bit* is set to 1 a Controlled-R. In this case the reversible XOR is equivalent to a Controlled-NOT, or "C-NOT".

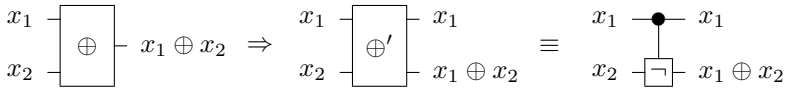

Figure 7: It is possible to make the XOR gate ($\oplus$) reversible by adding an extra output bit. We then obtain a C-NOT gate that performs a NOT ($\neg$) on the target bit x_2 if and only if the control bit x_1 is set to 1.

Adding adequate extra output bits to each irreversible gate, one can make a whole circuit reversible, but this will introduce a lot of extra bits that are useless for the computation but necessary for preventing dissipation. Indeed, it is the erasure of these "garbage" bits that would cause irreversibility and thus energy consumption. At first view, it may seem that this erasure would be necessary to free some memory space in order to perform subsequent computations, otherwise garbage bits would accumulate after each computation and finally saturate the computer memory. Actually this is not true as it may be shown that it is possible to arrange the garbage bits in such a way that they all end up in a standard state, for instance all bits with value 0, ready for a new computation (this will be even more crucial to quantum computation where extra bits in different states could destroy the interferences that are at the basis of most quantum algorithms). This means that it is theoretically really possible to perform computations with arbitrarily low energy consumption [Ben89].

Following these ideas, we may define a model of computation that involves reversible circuits only. As the translation of an irreversible circuit into a reversible one is always possible and may only increase the number of elementary gates and the work space polynomially, a crucial consequence is that this model will be computationally equivalent to its irreversible counterpart

and thus the complexity classes will remain unchanged. The basic difference is that reversible circuits will be built from a different set of universal gates, all being reversible. More precisely, it may be shown that to perform universal reversible computation, one needs a set that contains at least one *three-bit* gate (while it was only a *two-bit* gate for irreversible computation, for instance NAND). One example of universal set is made of only one such gate, the Fredkin gate [FT82]. This gate swaps the first two bits if the third bit is set to 1 and leaves them alone otherwise, so it may be considered as a Controlled-SWAP (see Fig. 8). It is indeed straightforward to check that

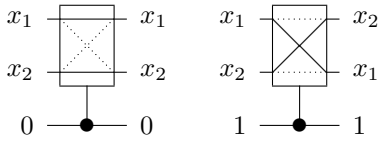

Figure 8: The Fredkin gate, which acts as a Controlled-SWAP: it swaps the first two bits if the third bit is set to 1 and leaves them alone otherwise.

this gate alone may reproduce the NOT, AND, OR, SWAP and FANOUT gates, with the help, when necessary, of extra bits prepared in a standard state, called *ancillae*, and producing garbage bits. Another interesting universal set is made of the Toffoli gate alone, also known as the "Controlled-Controlled-NOT" as it flips the third bit (the *target* bit) if the first two bits (the *control* bits) are both set to 1 and leaves it alone otherwise. This gate will play an important role in quantum computation, another even more physically based model of computation that we consider next.

2 Quantum computation

2.1 Quantum circuits

Following the lines of the definition of the circuit model of *classical* computation, we may now introduce the circuit model of *quantum* computation[1].

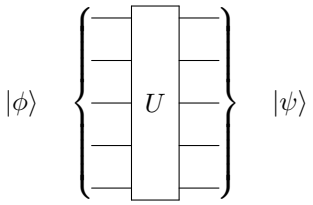

Figure 9: A quantum circuit implements a unitary operation U that transforms an input state $|\phi\rangle$ into an output state $|\psi\rangle$.

While a classical circuit is made of wires carrying the value of bits between logical gates, a quantum circuit consists of wires carrying qubit states between quantum gates. We have already defined the notion of qubit, but what is a quantum gate? Intuitively, it should be an application that maps input qubits to output qubits. As any physical transformation is reversible, the number of output qubits must match the number of input qubits, and therefore a gate will just act as a reversible transformation on a fixed number of qubits. For instance, it is clear that any reversible classical gate may be generalized to a quantum gate, for instance the NOT gate:

$$U_\neg : \mathcal{H}_2 \to \mathcal{H}_2 : |x_1\rangle \mapsto |\neg x_1\rangle, \tag{7}$$

[1]Let us recall that we have already mentioned in the introduction another model of quantum computation, the Quantum Turing Machine. As it is the case for their classical counterparts, these two quantum models are equivalent, but we prefer to focus on the circuit model which is more appropriate to design algorithms.

or the C-NOT gate (reversible XOR):

$$U_{\oplus} : \mathcal{H}_2 \otimes \mathcal{H}_2 \to \mathcal{H}_2 \otimes \mathcal{H}_2 : |x_1\rangle \otimes |x_2\rangle \mapsto |x_1\rangle \otimes |x_1 \oplus x_2\rangle. \tag{8}$$

This shows that reversible classical computation is a special case of quantum computation. However, quantum computation is more general as there are quantum gates that have no classical equivalent. Indeed, this will be the case for any quantum gate that produces superposition of computational basis states or introduces relative phases. More specifically, the evolution postulate only imposes that every physical evolution is unitary, so that a general quantum gate on n qubits will be described by a unitary operator on a Hilbert space of dimension $N = 2^n$. All these unitary operators form a compact Lie group generally noted $U(N)$. To perform universal quantum computation, we should therefore be able to simulate any element of $U(N)$ from a finite set of basic gates.

To better understand the implications of this requirement, let us first consider the simplest unitary group, that is, $U(1)$. This is isomorphic to the set of complex numbers with unit modulus, that is, numbers of the form $e^{i\phi}$ for any $\phi \in \mathbb{R}$. Therefore, we see that even in the simplest case of $U(1)$, we have an infinite number of group elements which form a continuum. Does it make it impossible to define a finite set of group elements that can generate the whole group? Not really, at least if we are happy with being able to approximate any group element within arbitrary precision. Indeed, let us consider a single group element $e^{i\phi}$ where the phase ϕ is such that $\phi/2\pi$ is irrational ($\phi/2\pi \in \mathbb{R} \setminus \mathbb{Q}$). Then, any group element $e^{i\phi'}$ can be approximated with arbitrarily small error by some power $(e^{i\phi})^k$. The same idea will be used to approximate the group $U(N)$ for larger N.

Let us now consider one-qubit gates, corresponding to the group $U(2)$. In the computational basis $(|0\rangle, |1\rangle)$, each gate U may be written as a unitary operator

$$U : \mathcal{H}_2 \to \mathcal{H}_2 : \begin{cases} |0\rangle & \mapsto & u_{00}|0\rangle + u_{10}|1\rangle \\ |1\rangle & \mapsto & u_{01}|0\rangle + u_{11}|1\rangle, \end{cases} \tag{9}$$

or equivalently as a unitary matrix

$$U = \begin{pmatrix} u_{00} & u_{01} \\ u_{10} & u_{11} \end{pmatrix}, \tag{10}$$

where $u_{kl} = \langle k|U|l\rangle$ are complex numbers satisfying the unitarity conditions $\sum_{k=0,1} u^*_{kl} u_{km} = \delta_{lm}$, that is, $U^\dagger U = I$. It follows that up to an irrelevant global phase, each one-qubit gate may be represented by a matrix of the following form

$$U_{(\theta,\phi)} = \begin{pmatrix} \cos\frac{\theta}{2} & -e^{i\phi}\sin\frac{\theta}{2} \\ e^{-i\phi}\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix}, \tag{11}$$

While classically there exist only two reversible one-bit gates, NOT($\neg$) and Identity, which can be generalized to the quantum gates $U_\neg$ and I as

$$U_\neg = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} = iU_{(\pi,\pi/2)} \qquad I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = U_{(0,0)}, \tag{12}$$

we see that there exists a whole continuum of one-qubit quantum gates, for all values of the angles θ and ϕ.

When studying universal set of quantum gates, we will always assume that the SWAP gate, which swaps two qubits, is implicitly included. In the case of classical computation, another gate was usually implicitly included: the FANOUT gate. However, we will not include it in our quantum model, simply because it is unphysical, as follows from the no-cloning theorem! Indeed, suppose we would like to define a quantum equivalent to the FANOUT classical gate. To ensure unitarity, it should be a two qubit gate, that copies an unknown qubit onto a *blank* qubit, that is a second

qubit prepared in a standard state, for instance $|0\rangle$. The quantum FANOUT U_{FO} would therefore map the computational basis states as

$$U_{\mathrm{FO}}|0\rangle \otimes |0\rangle = |0\rangle \otimes |0\rangle \quad (13)$$
$$U_{\mathrm{FO}}|1\rangle \otimes |0\rangle = |1\rangle \otimes |1\rangle, \quad (14)$$

effectively copying the first qubit onto the second (blank) one. However, due to the linearity of quantum mechanics, this gate would map the superposition state $\alpha|0\rangle + \beta|1\rangle$ as

$$U_{\mathrm{FO}}(\alpha|0\rangle + \beta|1\rangle) \otimes |0\rangle = \alpha|0\rangle \otimes |0\rangle + \beta|1\rangle \otimes |1\rangle \quad (15)$$

instead of

$$(\alpha|0\rangle + \beta|1\rangle) \otimes (\alpha|0\rangle + \beta|1\rangle), \quad (16)$$

and would therefore not copy such a superposition state reliably. More precisely, the no-cloning theorem of quantum mechanics implies that it is impossible to copy an unknown quantum state with arbitrarily high fidelity. Nonetheless, even though the FANOUT gate is an essential tool for classical computation, we will see that it is possible to design efficient quantum algorithms without the need to effectively copy qubits.

2.2 Universal set of quantum gates

Remember that a universal set for reversible classical computation consists in the Toffoli gate alone. As shown by Deutsch [Deu89], a very close quantum cousin to the Toffoli gate, which was later called to Deutsch gate, is universal for quantum computation. The Deutsch gate can be described as a Controlled-Controlled-R gate (recall that the Toffoli gate was a Controlled-Controlled-NOT, as shown in Fig. 10), where $R = iU_{(\theta,\pi/2)}$ for some phase θ such that θ/π is irrational. Let us give a sketch of the proof that this gate is universal. First note that R can be written as $R = i\exp(-i\frac{\theta}{2}\sigma_x)$, where σ_x is one of the Pauli matrices

$$\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad \sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \quad \sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

We will say that σ_x generates R. Therefore, the successive powers of R can approximate any matrix that diagonalizes in the same basis as σ_x, in particular the NOT gate which coincides with σ_x. Since a power of R can approximate NOT a power of Controlled-Controlled-R, i.e., the Deutsch gate, can approximate a power of Controlled-Controlled-NOT, i.e., the Toffoli gate. The Toffoli gate swaps the last two computational basis states of two-qubits: $|110\rangle \leftrightarrow |111\rangle$. Combining the Toffoli gate with the SWAP gate, we can actually swap any two computational basis states. Therefore, we are able to apply any unitary approximated by R (or, equivalently, and matrix $\exp(i\theta\sigma_x)$ generated by σ_x) on any two elements of the computational basis. Finally, by combining unitaries generated by σ_x on different states, we can also obtain unitaries generated by σ_y and σ_z, and in turn the whole group $SU(N)$, which corresponds to the group $U(N)$ up to a global phase which is irrelevant for quantum computation.

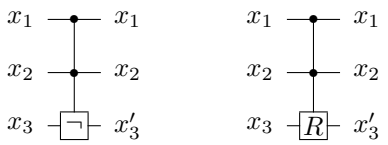

Figure 10: The Toffoli and Deutsch gates, which taken alone are universal respectively for reversible classical computation and quantum computation.

While all these universal sets involve three-bit (or three-qubit) gates, and it is indeed possible to show that universal reversible classical computation requires such gates, it is actually not necessary

for quantum computation. Indeed, one can simulate a Controlled-Controlled-R^2 gate using two Controlled-R, one Controlled-$R^\dagger$ and two C-NOT. Since a Controlled-R gate alone can simulate all these gates, this two qubit gate is by itself universal for quantum computation.

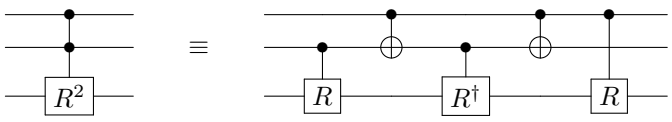

Figure 11: Simulating a Controlled-Controlled-R^2 from two Controlled-R, one Controlled-$R^\dagger$ and two C-NOT.

Finally, it can be shown that a generic 2-qubit gate, together with the SWAP gate, can approximate all 2-qubit gates, and therefore in particular a Controlled-R gate. This means that any generic 2-qubit gate is universal for quantum computation. Note that this is excellent news for practical implementations as it means that as soon as we are able to apply a generic interaction between two qubits, we are ready for universal quantum computation [DBE95, Llo95].

2.3 Quantum algorithm

We may now explicitly define the notion of a quantum algorithm. A quantum algorithm consists in

1. the preparation of the register in a quantum state of n qubits taken from the computational basis (for instance a state of the form $|x\rangle|0\rangle|0\rangle\cdots|0\rangle$, where x is the input to the problem),
2. a quantum circuit C_n made of quantum gates taken from a universal set for quantum computation,
3. a projective measurement of the output register in the computational basis.

Practically, the circuit C_n will depend on the size of the particular instance of the problem it is supposed to solve through the number of needed qubits n in the quantum register. A quantum algorithm then actually corresponds to a family of circuits C_n for all values n. As in the classical case, we must add a *uniformity condition* to derive a proper notion of complexity, meaning that the design of circuit C_n may be built in time at most polynomial in n. The complexity of an algorithm is then defined as the size of the circuit C_n (as for classical circuits, the size of C_n is the number of gates in C_n).

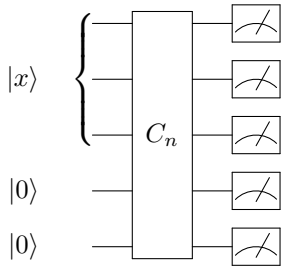

Figure 12: Schematic description of a quantum algorithm, where C_n is a quantum circuit made of universal gates.

These definitions deserve a few further comments. First of all, one could think that this definition of quantum algorithms is rather restrictive and that it could be interesting to make use of quantum processes that do not fit in this definition.

For instance, it could be possible to prepare the quantum register in a state that does not belong to the computational basis, or similarly measure this register in another basis. However, such generalized algorithms could easily be translated into our more restrictive definition just by adding some gates at the beginning and the end of the quantum circuit which rotate the generalized basis into the computational one or vice-versa. Similarly, the use of a *projective* measurement does not induce a loss of generality as we have seen that any measurement, including the most general POVM-type, may be reduced to a projective measurement with the help of ancillae.

Furthermore, one could design quantum algorithms with intermediate measurements instead of only one final measurement. Nonetheless, it is possible to show that this kind of algorithm could also be translated into our model, just by postponing any measurement to the end of the algorithm. This is true even when the subsequent computation depends on the outcome of the intermediate measurements, although in such a case ancillae are needed. The trick is simply to perform the subsequent computation conditionally to these ancillae that will only be measured at the end of the algorithm.

2.4 Quantum complexity

Recall that we have defined the complexity of a quantum algorithm as the size of the circuit C_n, that is, the number of elementary gates in the circuit. However, a peculiarity of quantum computation is that a universal set of quantum gates may only simulate an arbitrary quantum gate up to some error. Since these errors could accumulate as the size of the circuit C_n increases with n, one could wonder if all universal sets will yield an equivalent notion of complexity when errors are taken into account. Fortunately, it is indeed the case because we are only interested in the overall scaling of the complexity. More precisely, we will consider an algorithm efficient if it can solve a problem with a probability higher that $2/3$ and a complexity that grows at most polynomially in n. Similarly to the classical class BPP, we then define the quantum class BQP, for *bounded error quantum polynomial*, that includes any problem that may be solved by such an efficient algorithm.

Let us consider a quantum algorithm made of a series of gates $U_1, U_2, \ldots, U_t$. Suppose now that we want to apply this algorithm using a universal set of gates that can only approximate each gate U_t up to some error ϵ, more precisely we can obtain an approximation $\tilde{U}_t$ such that $\|\tilde{U}_t - U_t\| \leq \epsilon$, where we have used the operator norm

$$\|\tilde{U}_t - U_t\| = \max_{|\psi\rangle} \|(\tilde{U}_t - U_t)|\psi\rangle\|,$$

the maximum being taken over normalized states $|\psi\rangle$. Let us compare the intermediate states of the ideal algorithm $|\phi_t\rangle = U_t|\phi_{t-1}\rangle$, where $|\phi_0\rangle$ is the input state, with the intermediate state of the algorithm that we can implement, that is, $\left|\tilde{\phi}_t\right\rangle = \tilde{U}_t\left|\tilde{\phi}_{t-1}\right\rangle$, where $|\phi_0\rangle = \left|\tilde{\phi}_0\right\rangle$. Defining the error state $|E_t\rangle = (\tilde{U}_t - U_t)\left|\tilde{\phi}_{t-1}\right\rangle$, we have

$$\begin{aligned}
\left|\tilde{\phi}_1\right\rangle &= \tilde{U}_1|\phi_0\rangle = U_1|\phi_0\rangle + (\tilde{U}_1 - U_1)|\phi_0\rangle = |\phi_1\rangle + |E_1\rangle \\
\left|\tilde{\phi}_2\right\rangle &= \tilde{U}_2\left|\tilde{\phi}_1\right\rangle = U_2\left|\tilde{\phi}_1\right\rangle + (\tilde{U}_2 - U_2)\left|\tilde{\phi}_1\right\rangle = |\phi_2\rangle + |E_2\rangle + U_2|E_1\rangle \\
&\vdots \\
\left|\tilde{\phi}_T\right\rangle &= |\phi_T\rangle + |E_T\rangle + U_T|E_{T-1}\rangle + U_T U_{T-1}|E_{T-2}\rangle + \ldots + U_T U_{T-1} \ldots U_2|E_1\rangle.
\end{aligned}$$

Finally, since for any unitary V, we have $\|V|E_t\rangle\| = \||E_t\rangle\|$, we have by the triangle inequality

$$\|\left|\tilde{\phi}_T\right\rangle - |\phi_T\rangle\| \leq \||E_T\rangle\| + \||E_{T-1}\rangle\| + \ldots + \||E_1\rangle\| \leq T \cdot \epsilon,$$

where we have used the fact that $\||E_t\rangle\| = \|(\tilde{U}_t - U_t)\left|\tilde{\phi}_{t-1}\right\rangle\| \leq \|\tilde{U}_t - U_t\|$ (by definition of the operator norm).

Therefore, in order to keep the error below some constant, we need to approximate the gates within error $\epsilon = O(1/T)$. Moreover, it may be shown that the error introduced when simulating any quantum gate by a universal set may be reduced to ϵ using a circuit of elementary gates whose size grows polynomially in $O(1/\epsilon)$ only. It implies that compensating the accumulation of errors along a circuit only introduces a polynomial overhead and therefore does not affect the definition of the class BQP. Moreover, it also implies that the definition of the class BQP does not depend on the choice of universal set, since any universal set will be able to approximate any other universal set with only polynomial overhead.

How does the class BQP compare to classical classes of complexity? First, since any classical reversible gate is a special case of a quantum gate, quantum computation is necessarily as powerful as reversible computation, which is itself universal for classical computation, meaning that $P \subseteq BQP$. Moreover, it is clear that a quantum circuit can simulate a random coin flip by preparing a qubit in the state $|0\rangle$, applying the Hadamard gate H to obtain the state $\frac{1}{\sqrt{2}}[|0\rangle + |1\rangle]$, and measuring this state in the computational basis (as already discussed, this measurement can in turn be postponed until the end of the algorithm using an ancilla qubit). Therefore, since quantum computation has the power of P and can simulate random coin flips, it has the power of BPP, that is, we have $BPP \subseteq BQP$.

On the other side of the complexity spectrum, how powerful can quantum computation be? It can be shown that any quantum computation can be simulated on a classical computer (consistently with the Church-Turing thesis), with possibly exponential overhead in time, but only polynomial overhead in space. This implies that $BQP \subseteq PSPACE$. Indeed, let us consider a quantum algorithm consisting in the application of a unitary $U = U_t U_{t-1} \cdots U_2 U_1$ (where the U_i's are the gates constituting the circuit) on an initial state $|x\rangle$. Then, the probability to obtain the measurement outcome y is given by $p_y = |\langle y|U|x\rangle|^2$, where the amplitude may be expanded as:

$$\langle y|U|x\rangle = \sum_{x_1,\cdots,x_{t-1}} \langle y|U_t|x_{t-1}\rangle\langle x_{t-1}|U_{t-1}|x_{t-2}\rangle \cdots \langle x_2|U_2|x_1\rangle\langle x_1|U_1|x\rangle$$

Even though this sum contains an exponential number of terms (more precisely $2^{n(t-1)}$ terms), each of these terms can be evaluated using polynomial space, and therefore the whole sum can be evaluated progressively without using more than polynomial space. In conclusion, we have the following overall picture:

$$P \subseteq BPP \subseteq BQP \subseteq PSPACE$$

The main motivation for quantum computation is that there could be problems which can be solved efficiently on a quantum computer but on a classical computer. In terms of complexity classes, this would translate into the fact that BQP is strictly larger than BPP. However, this statement is still unproven (it would actually also imply important consequences for classical complexity theory, since it would resolve the conjecture $P \neq PSPACE$). Nevertheless, there is already some evidence that quantum computers are more powerful that their classical analogues, as witnessed by different quantum algorithms providing various speed-ups over classical algorithms. These can be classified in three different types of speed-ups (which will be later studied in this course):

- Non-exponential speed-ups. This is the case of Grover's algorithm for unstructured search [Gro96], which provides a quadratic speed-up over classical algorithms.
- Relativized exponential separation. In this case, it is shown that given access to some black-box operation O, often called oracle, a quantum algorithm can be exponentially more powerful than any classical algorithm. This is the case of Simon's algorithm for period finding [Sim94], which implies that $BPP^O \neq BQP^O$, but not the *unrelativized* analogue $BPP \neq BQP$
- Exponential speed-up over the best *known* classical algorithm. This is the case of Shor's quantum algorithm for FACTORING [Sho94]. This algorithm implies that FACTORING is

in BQP, and while it is sill unknown whether this problem belongs to the class BPP, as of now it has not been possible to prove that no efficient classical algorithm exists for this problem.

References

[AKS04] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. PRIMES is in P. *Annals of Mathematics*, 160(2):pp. 781–793, 2004.

[Ben73] Charles H. Bennett. Logical reversibility of computation. *IBM J. Res. Dev.*, 17(6):525–532, 1973.

[Ben89] Charles H. Bennett. Time-space trade-offs for reversible computation. *SIAM J. Comput.*, 18:766–776, 1989.

[Chu36] Alonzo Church. An unsolvable problem of elementary number theory. *Am. J. Math.*, 58:345, 1936.

[Cob65] Alan Cobham. The intrinsic computational difficulty of functions. In *Proceedings of the International Conference on Logic, Methodology, and Philosophy of Science*, pages 24–30, 1965.

[DBE95] David Deutsch, Adriano Barenco, and Artur Ekert. Universality in quantum computation. *Proc. R. Soc. London A*, 449:669–677, 1995.

[Deu89] David Deutsch. Quantum computational networks. *Proc. R. Soc. London A*, 425:73, 1989.

[Edm65] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17(3):449–467, 1965.

[FT82] Edward Fredkin and Tommaso Toffoli. Conservative logic. *Int. J. Theor. Phys.*, 21(3/4):219–253, 1982.

[Göd31] Kurt Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. *Monatshefte für Mathematik*, 38(1):173–198, 1931.

[Gro96] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the 28th Annual Symposium on the Theory of Computing*, pages 212–219, New York, 1996. ACM Press.

[Hil02] David Hilbert. Mathematical problems. *Bulletin of the American Mathematical Society*, 8(10):437–479, 1902.

[Lan61] Rolf Landauer. Irreversibility and heat generation in the computing process. *IBM J. Res. Dev.*, 5:183, 1961.

[Llo95] Seth Lloyd. Almost any quantum logic gate is universal. *Phys. Rev. Lett.*, 75(2):346, 1995.

[Sho94] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. In S. Goldwasser, editor, *Proceedings of the 35th Annual Symposium on the Foundations of Computer Science*, pages 124–134, New York, 1994. IEEE Computer Society Press. e-print quant-ph/9508027.

[Sim94] Daniel R. Simon. On the power of quantum computation. In S. Goldwasser, editor, *Proceedings of the 35th Annual Symposium on the Foundations of Computer Science*, pages 116–123, New York, 1994. IEEE Computer Society Press.

[SS77] R. Solovay and V. Strassen. A fast Monte-Carlo test for primality. *SIAM Journal on Computing*, 6(1):84–85, 1977.

[Tur36] Alan M. Turing. On computable numbers, with an application to the Eintscheidungsproblem. *Proc. Lond. Math. Soc. 2*, 42:230, 1936.