Quantum Algorithms

– Lecture Notes*–

Summer School on Theory and Technology in Quantum Information, Communication, Computation and Cryptography

Julia Kempe†
CNRS & LRI, Université de Paris-Sud
91405 Orsay, France

June 2, 2006

*These lecture notes are based on a book chapter written by the author for "Lectures in Quantum Information", edited by D. Bruss and G. Leuchs and to be published by Birkhäuser in 2006

†kempe@lri.fr

Contents

1 Introduction

The idea to use quantum mechanics for algorithmic tasks may be traced back to Feynman [24, 25]. The application he had in mind was the simulation of quantum mechanical systems by a universal quantum system, the quantum computer. Feynman argued that quantum mechanical systems are well equipped to simulate other quantum mechanical systems; hence a universal quantum machine might be able to efficiently do such simulations. Another approach to this question was taken by Deutsch [14], who tried to reconcile quantum mechanics and the Church-Turing principle, which (roughly speaking) states that any computable function can be calculated by what is known as a *universal Turing machine*. Deutsch put the notion of a universal machine on a physical footing and asked if the principle had to be modified if the machine was quantum mechanical, establishing what has since been

known as the *Church-Turing-Deutsch principle*. In his work Deutsch was also the first to exhibit a concrete computational task, which is impossible to solve on a classical computer yet which has an easy quantum mechanical solution, *Deutsch's algorithm* (see next section). What is interesting about this algorithm is not only that it is the smallest algorithm, involving only two quantum bits (qubits). It also carries the main ingredients of later quantum algorithms, and is a nice toy model for understanding why and how quantum algorithms work.

A major breakthrough in quantum algorithms was made in 1994 by Peter Shor, who gave an efficient quantum factoring algorithm. Factoring numbers into primes is an important problem, and no efficient classical algorithm is known for it. In fact many cryptographic systems rely on the assumption that factoring and related problems, like discrete logarithm, are hard problems. Shor's algorithm has put a threat on the security of many of our daily transactions - should a quantum computer be build, most current encryption schemes will be broken immediately.

In these lectures we will introduce quantum circuits and describe some quantum algorithms leading up to Shor's celebrated algorithm for factoring. We will also give the details of another notable quantum algorithm: Grover's 1996 algorithm for unstructured search. However, quantum algorithm design has not stopped with Shor's discovery, and in the last few chapters of these notes we will describe some generalizations of Shor's algorithm (the Hidden Subgroup problem) as well as some new algorithms and algorithmic techniques.

Those who are interested to know more will find detailed expositions of the classic quantum algorithms in the literature (in particular [41, 37]) and of more recent developments in the extensive reference list at the end of these notes.

2 Notation

In these lectures we will make use of the following conventions and notations:

A *bit* b is either 0 or 1, i.e. $b \in \{0,1\}$. An *n-bit string* is a sequence of n bits, i.e. it is an element of $\{0,1\}^n$. We often work in arithmetic over the 2-element field $GF(2)$. In this case we denote addition by $\oplus$: $0 \oplus 0 = 1 \oplus 1 = 0$ and $0 \oplus 1 = 1 \oplus 0 = 1$, this is just addition mod 2. We will also use $\oplus$ to denote bit-wise addition of bit strings mod 2, e.g. $10 \oplus 11 = 01$. For multiplication of two bits b_1 and b_2 we will write either $b_1 b_2$ or $b_1 \cdot b_2$. The *inner product* of two bit strings of same length n will also be denoted by "$\cdot$" and is defined

for $x = x_1x_2 \ldots x_n$ and $y = y_1y_2 \ldots y_n$ as $x \cdot y = x_1y_1 + x_2y_2 + \ldots + x_ny_n$ mod 2.

A qubit is a two-dimensional quantum system. We will denote the two basis states by $|0\rangle$ and $|1\rangle$. The state of several qubits is spanned by the *tensor product* of individual qubits. If the first qubit is in the state $|\phi\rangle$ and the second qubit is in the state $|\psi\rangle$, then we will write the state of the joint system of two qubits as $|\phi\rangle \otimes |\psi\rangle$. In case there is no ambiguity we will omit the tensor product $\otimes$. The space of n qubits is spanned by the 2^n states $|00\ldots00\rangle$, $|00\ldots01\rangle$, ..., $|11\ldots11\rangle$. These states are also called *computational basis* states, as they correspond to the classical configurations of n bits. A *measurement* in the computational basis projects onto these basis states. Its outcomes are hence described by an n-bit string. If the system is in a state $|\phi\rangle$, then the probability that a measurement in the computational basis gives a bit string x is given by the inner product squared $|\langle x|\,\phi\rangle|^2$. Sometimes we will only perform a *partial* measurement on a state. This means we only measure a subset of the qubits in the computational basis. The state collapses into a state that is consistent with the measurement we have performed. For instance if we have the state $\frac{1}{\sqrt{3}}|00\rangle + \frac{1}{\sqrt{3}}|01\rangle + \frac{1}{\sqrt{3}}|11\rangle$ and we measure the second qubit in the computational basis, then the state of the first qubit will be $|0\rangle$ if the outcome is 0 (which happens with probability $\frac{1}{3}$) and $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ if we get outcome 1 (with probability $\frac{2}{3}$). This becomes more obvious if we rewrite the state as

$$\frac{1}{\sqrt{3}}|00\rangle + \frac{\sqrt{2}}{\sqrt{3}}\frac{|0\rangle + |1\rangle}{\sqrt{2}} \otimes |1\rangle\,.$$

To analyse the algorithms that follow, we will need some standard notation to describe the asymptotic behavior of such functions like the running time of an algorithm as a function of the input size. We will frequently use the following:

- $f = O(g)$ (f is "big-O" of g) if there exist positive constants C and k such that $|f(x)| \leq C|g(x)|$ for all $x > k$.
- $f = \Omega(g)$ (f is "big-Omega" of g) if there exist positive constants C and k such that $|g(x)| \leq C|f(x)|$ for all $x > k$.
- $f = \Theta(g)$ (f is "big-Theta" of g) if there exist positive constants C_1, C_2 and k such that $C_1|g(x)| \leq |f(x)| \leq C_2|g(x)|$ for all $x > k$.

3 The quantum circuit model

A classical computer can be described by a circuit. The input is a string of bits ($\{0,1\}^n$). The input is processed by a succession of logical gates like NOT, OR, AND or NAND, which transforms the input to the output. In general the output bits are Boolean functions $f : \{0,1\}^n \to \{0,1\}$ of the input bits. A schematic view is depicted in Fig. 1.

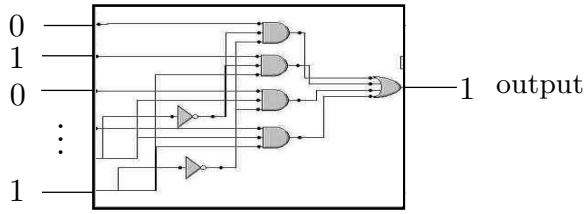

Figure 1: Schematic representation of a classical circuit computing a Boolean function.

The way Feynman put it, a quantum computer is a machine that obeys the laws of quantum mechanics, rather than Newtonian classical physics. In the context of computation this has two important consequences, which define the two aspects in which a quantum computer differs from its classical counterpart. First, the *states* describing the machine in time are quantum mechanical wave functions. Each basic unit of computation - the *qubit* - can be thought of as a two-dimensional complex vector of norm 1 in some Hilbert space. The two dimensional basis for such a qubit is often labelled as $|0\rangle$ and $|1\rangle$, where the basis states correspond to the classical bit. And second, the *dynamics* that governs the evolution of the state in time is *unitary*, i.e., described by a unitary matrix that transforms the state at a certain time to the state at some later time. A second dynamical ingredient is the *measurement*. In quantum mechanics observing the system changes it. In the more restricted setting of a quantum algorithm a measurement can be thought of as a projection on the basis states. A particular basis state will be measured with a probability which is given by the squared amplitude in the state that is being measured.

With these notions in place, we can describe a general quantum circuit. In the beginning the qubits are initialized to some known classical state (a basis state of $|0\rangle$ and $|1\rangle$, also called the *computational basis*), like for instance $|0\rangle \otimes |0\rangle \otimes \ldots \otimes |0\rangle$, which we will denote by $|00\ldots0\rangle$. Then a unitary transformation U is performed on the qubits. In the end the qubits are measured in the computational basis and the result is processed classically.

The general setting can be seen in Fig. 2.

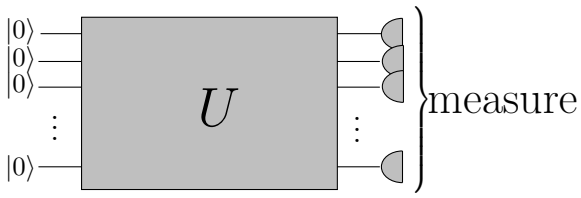

Figure 2: A general quantum circuit consisting of three steps: initialization of the qubits, unitary transformation and measurement in the computational basis.

Given this model, it is not even clear if such a quantum computer is able to perform classical computations. After all, a unitary matrix is invertible and hence a quantum computation is necessarily *reversible*. Classical computation given by some circuit with elementary gates, like the AND and NOT gate, is not reversible, let alone because a gate like the AND gate has two inputs and only one output. However, the question of reversibility of classical computation has been studied in the context of energy dissipation by Bennett in the 70s [8] (see also [49]), who established that classical computation can be made reversible with only a polynomial overhead in the number of bits and gates used.

Universal Reversible Classical Computation One way to make classical computation reversible is to simulate any classical circuit with AND and NOT operations by a circuit consisting entirely of what is called a Toffoli gate, which acts as depicted in Fig. 3.

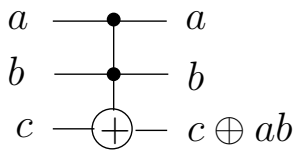

Figure 3: The Toffoli gate flips the last bit if the first two bits are in the state 11.

When viewed as a matrix over the 8 basis states 000, 001, 010, 011, 100,

101, 110, 111 the Toffoli gate acts as

$$T = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Clearly, the Toffoli gate is reversible (it is its own inverse and implements a permutation of the 8 bit strings).

To show how a circuit consisting of Toffoli gates can simulate *any* classical circuit we will assume here that the classical circuit is made of NAND gates and FANOUT gates. It is known (and not hard to show) that any Boolean function can be computed by a circuit of NAND and FANOUT gates. The FANOUT gate simply copies one bit into two bits. The NAND gate outputs 0 if and only if both input bits are 1; otherwise it outputs 1. In other words, it acts as $\mathrm{NAND}(a, b) = 1 \oplus ab$. Fig. 4 shows how the Toffoli gate can implement the FANOUT and the NAND gate with some extra ancillary bits.

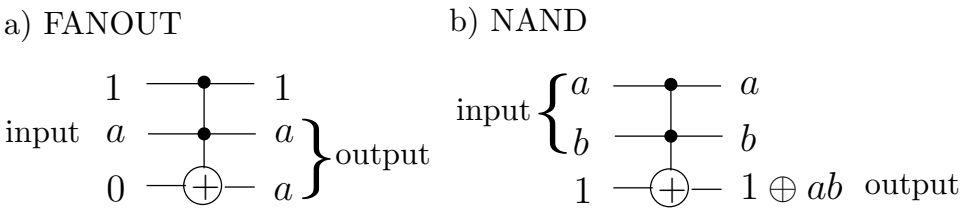

Figure 4: Implementing the FANOUT (a) and the NAND gate (b) with the Toffoli gate.

Replacing every occurence of a FANOUT or a NAND gate in a circuit by the corresponding Toffoli gate supplemented with ancillary bits, we can simulate the whole circuit in a reversible fashion. Since classical reversible computation is just a permutation on the bit strings of its input, it is in particular unitary. As a result, quantum computation is at least as strong as classical computation.

The next important question is whether it is possible to build a universal quantum machine (rather than special purpose computers). In other words,

is there a small set of operations that implements any unitary transformation? Classically, it is well known that any Boolean function can be computed with a small set of *universal* gates, like AND and NOT, or $NAND$, as we have seen above. Moreover each of the elementary gates in a classical circuit only operates on few (one or two) bits at a time. Fortunately, it turns out that a similar statement is true for the quantum world; it was shown [15, 17] that there are sets of universal quantum gates on at most 2 qubits. In particular it has been shown that any unitary transformation can be decomposed into a sequence of gates consisting of one-qubit gates and CNOTs. We can describe the action of the CNOT gate by showing how it transforms the basis states $|00\rangle$, $|01\rangle$, $|10\rangle$ and $|11\rangle$. The CNOT gate acts as in Fig. 5.

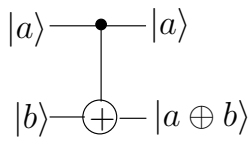

Figure 5: The CNOT gate flips the second qubit if the first qubit is in the state $|1\rangle$.

However, the set of gates consisting of CNOT and single qubit gates is a continuous gate set. Sometimes it is convenient to work with a small finite set of universal gates. One such universal gate set is the set of three gates $\{\frac{\pi}{8}, H, CNOT\}$, where

$$\frac{\pi}{8} = \begin{pmatrix} e^{\frac{-i\pi}{8}} & 0 \\ 0 & e^{\frac{i\pi}{8}} \end{pmatrix} \quad H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

$\frac{\pi}{8}$ is a gate that multiplies the basis state $|1\rangle$ with $e^{i\frac{\pi}{4}}$, the Hadamard gate H maps $|0\rangle \rightarrow \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $|1\rangle \rightarrow \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$ and the controlled NOT gate, CNOT, is as defined above. This gate set is universal in the sense that any unitary transformation can be *approximated* to arbitrary precision by a sequence of gates from this set.

It turns out that there are many other choices of finite universal gate sets acting on few qubits only. In principle the complexity of a unitary transformation, when counted as the number of elementary gates needed to implement it, might depend on the choice of universal gate set. However, Solovay and Kitaev have shown that all finite universal gate sets are equivalent in the sense that if we can approximate an n-qubit unitary with $poly(n)$ gates from one set, we can also approximate it with $poly(n)$ gates from the

other. In complexity theory, problems that can be solved by circuits with a number of gates polynomial in the size of the input are called *efficiently solvable*.

When we wish to solve a problem (like factor a large number) we are often content with a circuit that solves the problem *with some probability* p, which is not too small. If we have such a circuit and we can verify rapidly if the circuit gave the correct answer (in the case of factoring we just need to check if the numbers that are output are indeed factors of our original number), we can repeat it several times to boost the success probability. Repeating the circuit r times boosts the success probability up to $1-(1-p)^r$ which is exponentially close to 1.

In complexity theoretic terms, a problem that can be solved by a quantum circuits with $poly(n)$ elementary gates (from some fixed universal gate set) with high probability, where n is size of the input, is said to be in the complexity class BQP (which stands for bounded quantum polynomial time).

4 First Algorithms: Deutsch and Deutsch-Josza

As mentioned before, the first quantum algorithm is Deutsch's algorithm [14]. Before we describe it, let us clarify the notion of a quantum black-box function. A black box function is often used to model a subroutine of the calculate. We are then interested to know how often this subroutine needs to be performed to solve a problem. In this context we do not care about the details of the calculation in the subroutine, we simply model the function that is computed as a black box. Many of the separations between classical and quantum computing power will be formulated in the black box or *oracle* model. We will see that for certain problems a quantum algorithm needs to make substantially less calls - or *queries* - to the black box than any classical algorithm.

Classically, a black-box function can be simply thought of as a box that evaluates an unknown function f. The input is some n-bit string x and the output is given by an m-bit string $f(x)$. Quantumly, such a box can only exist if it is reversible. To create a reversible box, the input (x) is output together with $f(x)$ and the black box looks like in Fig. 6:

To make the box reversible, an additional m-bit input y is added and the output of the result is $f(x) \oplus y$ where $\oplus$ denotes bitwise addition mod 2. In particular, if y is fixed to be $y = 0 \ldots 0$, the output is $f(x)$. This reversible box, when given to a classical machine, is no stronger than the

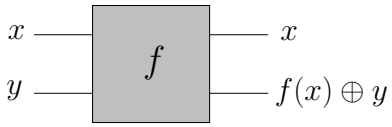

Figure 6: A reversible black box for a function $f : \{0,1\}^n \longrightarrow \{0,1\}^m$.

corresponding simple non-reversible box that maps x to $f(x)$. Note that this box now induces a transformation on $n+m$-bit strings that can be described by a permutation of the 2^{n+m} possible strings; in particular it is unitary.

Deutsch's algorithm With these notions in place we can give Deutsch's algorithm [14].

Problem: *Given a black-box function f that maps one bit to one bit, determine if the function is constant ($f(0) = f(1)$) or balanced ($f(0) \neq f(1)$).*

Note that classically, to solve this problem with a success probability bigger than one half, a machine has to query the black box twice; both $f(0)$ and $f(1)$ are needed. Deutsch's ingenuity is to use interference of the amplitudes of the quantum state such that only one query to the black box suffices. The following circuit on two qubits gives the quantum algorithm.

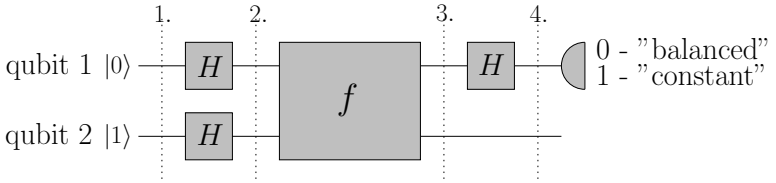

Figure 7: Deutsch's circuit.

1. The qubits are initialized in $|0\rangle\,|1\rangle$, the first ket denotes the qubit 1 and the second one qubit 2.

2. After the Hadamard transform is applied to each qubit, the state is $\frac{1}{2}\left(|0\rangle + |1\rangle\right)\left(|0\rangle - |1\rangle\right)$.

3. After the invocation of the black box the state of the two qubits is

$$\frac{1}{2}\left(|0\rangle\left(|f(0)\rangle - |f(0)\oplus 1\rangle\right) + |1\rangle\left(|f(1)\rangle - |f(1)\oplus 1\rangle\right)\right).$$

Note that the state of the second qubit in this expression is $\pm(|0\rangle - |1\rangle)$; the sign depends on the value of $f(0)$ (resp. $f(1)$). The state can be rewritten as

$$\frac{1}{2}\left((-1)^{f(0)}|0\rangle + (-1)^{f(1)}|1\rangle\right)(|0\rangle - |1\rangle).$$

4. After the last Hadamard is applied, the state of the first qubit becomes

$$\frac{1}{2}\left((-1)^{f(0)}(|0\rangle + |1\rangle) + (-1)^{f(1)}(|0\rangle - |1\rangle)\right),$$

which can be rewritten as

$$\frac{1}{2}\left((-1)^{f(0)} + (-1)^{f(1)}\right)|0\rangle + \left((-1)^{f(0)} - (-1)^{f(1)}\right)|1\rangle$$

. If the function is constant, this state is $\pm|0\rangle$, if it is balanced, the state is $\pm|1\rangle$. The final measurement will completely distinguish these two cases.

As a result, Deutsch's algorithm saves one query compared to the best possible classical algorithm for this problem. One query might seem very little, yet we will see how this algorithm has been generalized in several steps to ultimately factor numbers.

Deutsch-Josza algorithm In a first step, Deutsch and Josza [16] generalized Deutsch's algorithm to give a problem where the quantum algorithm gives more than just a single query advantage. It is, however, a *promise problem.*

Problem: *Given a black-box function f that maps n bits to one bit, with the promise that the function is constant ($f(x) = f(y)$) or balanced on exactly half the inputs (for all x there are exactly 2^{n-1} different y such that $f(x) \neq f(y)$), determine which one is the case.*

Note that classically, to solve this problem deterministically, one needs $2^{n-1}+1$ queries in the worst case, as in the balanced case one might have to query 2^{n-1} different y for some x before one finds a y such that $f(x) \neq f(y)$. The Deutsch-Josza algorithm solves this problem with one quantum query with the following algorithm. The analysis of this algorithm is very similar to Deutsch's algorithm. The difference in the circuit is that the Hadamard transform on one qubit is replaced with the tensor product of n Hadamard transforms $H^{\otimes n}$ on n qubits. Let us first analyze the action of $H^{\otimes n}$ on a

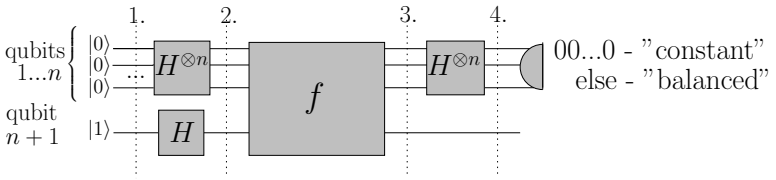

Figure 8: Deutsch-Josza algorithm.

basis state $|x\rangle$ (x is an n-bit string). The transformation induced by a single Hadamard on a qubit i in the basis state $|x_i\rangle$ can be written as

$$H|x_i\rangle = \frac{1}{\sqrt{2}}(|0\rangle + (-1)^{x_i}|1\rangle) = \sum_{y_i \in \{0,1\}} (-1)^{x_i \cdot y_i} |y_i\rangle$$

. Applying this to $H^{\otimes n}$ with $|x\rangle = |x_1 \dots x_n\rangle$ we get

$$\begin{aligned} & H^{\otimes n}|x_1 \dots x_n\rangle \\ = & \frac{1}{\sqrt{2^n}} \left(\sum_{y_1 \in \{0,1\}} (-1)^{x_1 \cdot y_1} |y_1\rangle \right) \otimes \dots \otimes \left(\sum_{y_n \in \{0,1\}} (-1)^{x_n \cdot y_n} |y_n\rangle \right) \\ = & \frac{1}{\sqrt{2^n}} \sum_{y \in \{0,1\}^n} (-1)^{x_1 \cdot y_1 + \dots + x_n \cdot y_n} |y\rangle = \frac{1}{\sqrt{2^n}} \sum_{y \in \{0,1\}^n} (-1)^{x \cdot y} |y\rangle , \quad (1) \end{aligned}$$

where $x \cdot y$ is the inner product of the vectors x and y mod 2. The Hadamard transform H and $H^{\otimes n}$ are instances of a more general transformation, called the Quantum Fourier transform (QFT). In our circuit $H^{\otimes n}$ gives in step 2. the state

$$\frac{1}{\sqrt{2^n}} \sum_{y \in \{0,1\}^n} |y\rangle \frac{1}{\sqrt{2}} (|0\rangle - |1\rangle) .$$

As before, at step 3. the state of the system on the first n qubits, is

$$|\phi_3\rangle := \frac{1}{\sqrt{2^n}} \sum_{y \in \{0,1\}^n} (-1)^{f(y)} |y\rangle .$$

If the function is constant, then this state is just the uniform superposition over all bit strings (up to a global phase) and using Eq. (1) we see that the state at step 4. is simply $(-1)^{f(0)}|0 \dots 0\rangle$. If the final measurement gives the all zero string the output of the algorithm is "constant". Otherwise the overlap of our state of the first n qubits at step 4., $H^{\otimes n}|\phi_3\rangle$, with the state $|0 \dots 0\rangle$ is 0, which means a measurement never gives the all zero string.

To see this, note that $\langle 0\ldots 0(H^{\otimes n}\,|\phi_3\rangle) = (\langle 0\ldots 0|H^{\otimes n})\,|\phi_3\rangle$ and let us calculate the inner product of $|\phi_3\rangle$ with the Fourier-transform of the all zero state:

$$\begin{aligned}\langle 0\ldots 0|H^{\otimes n}\big|\,\phi_3\rangle & \\ &= \frac{1}{2^n}\left(\sum_{y_1\in\{0,1\}^n}\langle y_1\,|\right)\left(\sum_{y_2\in\{0,1\}^n}(-1)^{f(y_2)}\,|y_2\rangle\right)\\ &= \frac{1}{2^n}\sum_{y_1\in\{0,1\}^n}(-1)^{f(y_1)}. \qquad (2)\end{aligned}$$

Using that f is balanced we get that this sum is 0. Hence in case that f is constant a measurement will always give the all zero string whereas in the balanced case we will always get an outcome different from all zeroes. This completes the analysis.

Note, that the speed-up achieved by the quantum algorithm from $O(2^n)$ queries to 1 query only holds if we compare with a classical deterministic machine. If the classical machine is allowed to be probabilistic, then the classical query complexity reduces to $O(1)$: If we query the function at random then in the balanced case each of the two function values will be seen with probability 1/2 and with very high probability we will see two different function values after a constant number of queries.

5 Period finding

So far we have seen separations between classical and quantum complexity that did not hold if we allowed for randomness. The following two algorithms will give the first separations that hold even with randomness. They both involve finding the period of some function. Indeed, most problems where a quantum computer excels exponentially over the best known classical algorithm are of the period finding flavor[1].

In the next algorithm a quantum computer solves a problem with an exponential speed-up over the best classical *probabilistic* machine.

5.1 Simon's algorithm

This algorithm of Simon [48] finds the "period" of a function.

[1] with the notable exception of certain quantum walk based separations, see e.g. [12]

Problem: *Given a function from n bits to n bits with the promise that there is an n-bit string $a \neq 0 \ldots 0$ such that for all x, y $f(x) = f(y)$ if and only if $y = x \oplus a$, find a.*

One can show that the best any classical probabilistic machine can do is to query elements at random until a collision is found. The probability of a collision for two randomly chosen elements is about 2^{-n}, and a slightly more elaborate analysis shows that the expected number of queries until a collision happens among the queried elements is $O(2^{n/2})$.

Interestingly, the quantum algorithm is very similar to the Deutsch-Josza algorithm with the difference that there are now $2n$ qubits as input to the black box and no Hadamard transforms on the second block of qubits, see Fig. 9. This circuit implements a special case of what is called Quantum Fourier Sampling.

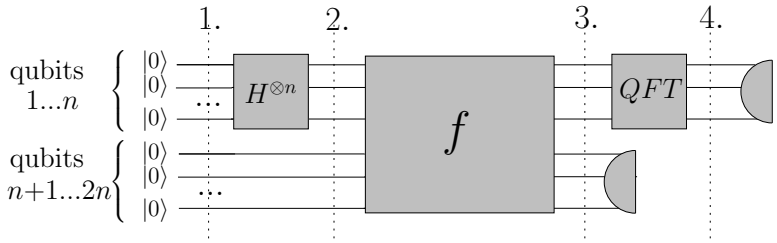

Figure 9: Simon's algorithm - Quantum Fourier Sampling. In our algorithm $QFT = H^{\otimes n}$. In general a QFT over a group G gives the Quantum Fourier Sampling algorithm over G.

Note that there is a partition of the 2^n input strings into two sets X and $\bar{X} = \{x \oplus a | x \in X\}$ with $|X|, |\bar{X}| = 2^{n-1}$, such that all the values $f(x)$ are distinct for $x \in X$ and similarly for $\bar{X}$. At step 3. the state is

$$\frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle |f(x)\rangle = \frac{1}{\sqrt{2^{n-1}}} \sum_{x \in X} \frac{1}{\sqrt{2}} \left(|x\rangle + |x \oplus a\rangle\right) |f(x)\rangle .$$

A measurement of the qubits in the second register will yield one of the 2^{n-1} values of $f(x)$ with equal probability and collapse the state of the first register to $\frac{1}{\sqrt{2}} \left(|x\rangle + |x \oplus a\rangle\right)$ for a random $x \in X$. At step 4. the state

becomes

$$\begin{aligned}
&\frac{1}{\sqrt{2^{n+1}}} \sum_{y\in\{0,1\}^n} \left((-1)^{x\cdot y} + (-1)^{(x\oplus a)\cdot y}\right) |y\rangle \\
&\qquad = \frac{1}{\sqrt{2^{n+1}}} \sum_{y\in\{0,1\}^n} (-1)^{x\cdot y} \left(1 + (-1)^{a\cdot y}\right) |y\rangle \\
&\qquad = \frac{1}{\sqrt{2^{n-1}}} \sum_{y: y\cdot a=0} (-1)^{x\cdot y} |y\rangle \,.
\end{aligned}$$

A measurement of the first register gives a random $y = y_1$ such that $a \cdot y_1 = 0$. We can now repeat this algorithm to obtain y_2 with $a \cdot y_2 = 0$, y_3 with $a \cdot y_3 = 0$ and so on. These y_i form a subspace of the n-dimensional vector space of all n-bit strings (over $GF(2)$). If among the y_i there are $n-1$ vectors that are linearly independent (i.e. such that they span a space of dimension $n-1$), then the equations $a \cdot y_i$ completely determine $a \neq 0$. But for each set of y_i that do not yet span a space of dimension $n-1$ the probability that the next y will be outside the space is at least $1/2$, because the space spanned by them contains at most 2^{n-2} out of the 2^{n-1} possible y's. Hence after $O(n)$ repetitions of the algorithm with a probability exponentially close to 1 we will have enough information to determine a.

5.2 Shor's Factoring Algorithm

Conceptually it is now only a small step from Simon's algorithm to Shor's algorithm for factoring. The first necessary observation is that in order to find a factor of a number, it is sufficient to solve a problem called period finding, the problem Shor's algorithm [47] actually solves:

Problem (Period Finding): *Given a function $f : \mathbb{Z} \to \mathbb{Z}$ and an integer N with the promise that there is a period $a \leq N$ such that for all x, y, $f(x) = f(y)$ if and only if $y \in \{x, x \pm a, x \pm 2a, \ldots\}$, find a.*

Reduction from Factoring to Period Finding Let us assume that we want to factor the number N. Once we have an algorithm that gives one factor q of N, we can restart the algorithm on q and N/q; we obtain all factors of N after at most $\log N$ iterations. Assume N is odd and not a power of a prime (both conditions can be verified efficiently and moreover in these cases it is easy to find a factor of N). First, we select a random $1 < y < N$ and compute $GCD(y, N)$ (this can be done efficiently using the Euclidean algorithm). If this greatest common divisor is larger than 1 we

have found a non-trivial factor of N. Otherwise, y generates a multiplicative group modulo N. This group is a subgroup of $\mathbb{Z}_N^*$, the multiplicative group modulo N. The order of this group is determined by the factors of N (and is unknown to us). The smallest integer a such that $y^a \equiv 1 \mod N$, known as the *order* of y, is the period of the function $f_y(x) = y^x \mod N$. This function can be viewed as a function over $\mathbb{Z}$.

Invoking now the period finding algorithm we can determine a. *If* a is even then $N|(y^{\frac{a}{2}}+1)(y^{\frac{a}{2}}-1)$. We know that $N \nmid (y^{\frac{a}{2}}-1)$ ($\frac{a}{2}$ is not the period of f_y), so *if* $N \nmid (y^{\frac{a}{2}}+1)$ then N must have a common factor with each of $(y^{\frac{a}{2}} \pm 1)$ and we can determine it by computing $GCD(N, y^{\frac{a}{2}}-1)$. It remains to be shown that with probability at least $1/2$ over the choice of y both conditions are satisfied, i.e., both a is even and $N \nmid (y^{\frac{a}{2}}+1)$. This can be shown using the Chinese Remainder Theorem (see e.g. [37, 41, 42]).

In what follows we focus on solving the period finding problem. We use essentially the same quantum circuit as in Simon's algorithm Fig. 9, namely Quantum Fourier Sampling with an appropriate definition of the Quantum Fourier transform.

Definition: The Quantum Fourier transform over $\mathbb{Z}_M$, the cyclic group of numbers mod M, implements the unitary

$$QFT : |x\rangle \longrightarrow \frac{1}{\sqrt{M}} \sum_{y \in \mathbb{Z}_M} \omega^{x \cdot y} |y\rangle ,$$

where $\omega = e^{\frac{2\pi i}{M}}$ is an M^{th} root of unity.

Notice that the QFT over $\mathbb{Z}_2$ is just the Hadamard transform on one qubit, and in general the transformation $H^{\otimes n}$ in Deutsch-Josza and in Simon's algorithms implements the QFT over the group $\mathbb{Z}_2^n$. The Fourier transform in that case is just a tensor product of single qubit unitaries. The ingenious part of Shor's algorithm is to show that the QFT over $\mathbb{Z}_M$ is also implementable efficiently, i.e., in time polynomial in $\log M$, by a quantum circuit.

Implementation of the QFT Note that the QFT implements an $M \times M$ unitary matrix with entries $\omega^{x \cdot y}$. A naive classical algorithm that computes each entry separately and then sums the appropriate rows to compute each of the amplitudes $\sum_y \omega^{x \cdot y}$ will require $O(M^2)$ steps. However, there is a well known trick to speed up the evaluation of all these sums: The classical Fast Fourier transform (FFT) takes only time $O(M \log M)$ for this task. For ease of presentation let us assume that $M = 2^n$. To evaluate $\omega^{x \cdot y} = \exp(\frac{2\pi i x \cdot y}{2^n})$,

let us expand x in binary notation $x = x_{n-1}2^{n-1} + x_{n-2}2^{n-2} + \cdots + x_1 2 + x_0$ and similarly for y. In the product $x \cdot y$ we can ignore all terms divisible by 2^n as they do not contribute to the exponent. Now

$$\frac{x \cdot y}{2^n} = y_{n-1}(.x_0) + y_{n-2}(.x_1 x_0) + y_{n-3}(.x_2 x_1 x_0) + \cdots + y_0(.x_{n-1}x_{n-2}\ldots x_0).$$

The terms in parenthesis are binary expansions, e.g. $.x_2x_1x_0 = x_2 2^{-1} + x_1 2^{-2} + x_0 2^{-3}$. The amplitude

$$\sum_{y \in \mathbb{Z}_M} \omega^{x \cdot y} = \left(\sum_{y_{n-1} \in \{0,1\}} e^{2\pi i y_{n-1}(.x_0)} \right) \cdots \left(\sum_{y_0 \in \{0,1\}} e^{2\pi i y_0 (.x_{n-1}x_{n-2}\ldots x_0)} \right)$$

can now be evaluated sequentially in time $O(\log M)$ for each of the M values of x.

Quantum parallelism improves this drastically. We can write

$$\begin{aligned} \frac{1}{\sqrt{M}} \sum_{y \in \mathbb{Z}_M} \omega^{x \cdot y} |y\rangle &= \frac{1}{\sqrt{2^n}} \left(|0\rangle + e^{2\pi i (.x_0)} |1\rangle \right) \otimes \left(|0\rangle + e^{2\pi i (.x_1 x_0)} |1\rangle \right) \otimes \\ &\quad \cdots \otimes \left(|0\rangle + e^{2\pi i (.x_{n-1}\ldots x_1 x_0)} |1\rangle \right). \end{aligned}$$

Fig. 10 shows a circuit that implements this transformation on $\mathbb{Z}_8$. The

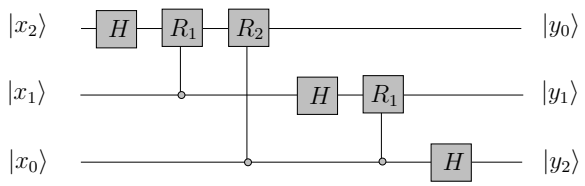

Figure 10: QFT on $\mathbb{Z}_8$. An element of $\mathbb{Z}_8$ is represented in binary notation $x = x_2x_1x_0$, $y = y_2y_1y_0$.

Hadamard on qubit x_i can be thought of as performing $|x_i\rangle \rightarrow (|0\rangle + e^{2\pi i(.x_i)} |1\rangle)$. The conditional rotations R_d give a phase of $e^{i\pi/2^d}$ to the qubit on which they act whenever the control qubit is in the state $|1\rangle$. The obvious generalization of this circuit to n qubits has $\frac{1}{2}n(n+1) = O(\log^2 M)$ gates.

Shor's algorithm for period finding With this implementation of the QFT in place we can analyse the algorithm in Fig. 9 for period finding. We need to chose the integer M over which the QFT is performed. For

our problem ($a \leq N$) we chose $M = 2^n$ to be a power of 2 such that $N^2 < M \leq N^4$.

For the moment, let us make the simplifying assumption that the period a divides M. At step 2. the first register is in a uniform superposition over all elements of $\mathbb{Z}_M$. As in Simon's algorithm the state at step 3. after the measurement of the second register is

$$\sqrt{\frac{a}{M}} \left(|x\rangle + |x+a\rangle + \ldots + \left| x + \left(\frac{M}{a} - 1 \right) a \right\rangle \right) |f(x)\rangle \tag{3}$$

for some random $x \in \mathbb{Z}_M$. The QFT transforms the state of the first n qubits into

$$\frac{\sqrt{a}}{M} \sum_{y \in \mathbb{Z}_M} \left(\sum_{j=0}^{M/a-1} \omega^{(x+ja)y} \right) |y\rangle = \frac{\sqrt{a}}{M} \sum_{y \in \mathbb{Z}_M} \omega^{xy} \left(\sum_{j=0}^{M/a-1} \omega^{jay} \right) |y\rangle .$$

Since a divides M, we have that whenever $\omega^{ay} \neq 1$, i.e., whenever $y \notin \{0, M/a, 2M/a, \ldots, (a-1)M/a\}$

$$\sum_{j=0}^{\frac{M}{a}-1} (\omega^{ay})^j = \frac{1-\omega^{My}}{1-\omega^{ay}} = 0.$$

This implies that in Eq. (4) the amplitudes of basis states $|y\rangle$ for y not a multiple of M/a are zero. Consequently the state at step 4. is a superposition over all $y \in \{0, M/a, 2M/a, \ldots, (a-1)M/a\}$ and a measurement gives a uniformly random $y = cM/a$. To extract information about a we need to solve $y/M = c/a$. Whenever c is coprime to a (which can be shown to happen with reasonable good probability $\Omega(1/\log\log a)$) we can write y/M as a minimal fraction; the denominator gives a.

In the (more likely case) that a does not divide M it is not hard to see that the same algorithm will give with high probability a y such that $|y/M - c/a| \leq 1/2M$ for some $0 \leq c < a$. But two distinct fractions with denominator at most N must be at least $1/N^2 > 1/M$ apart, so c/a is the unique fraction with denominator at most N within distance $1/2M$ from y/M and can be determined with the continued fraction expansion.

Note, that in Shor's algorithm the function $f_y(x) = y^x \mod M$ is not given by a black box, but needs to be computed every single time. This could be difficult since the exponent x is very large. However, using the binary expansion of x and repeated squaring, it is not hard to see that there exists a classical subroutine for computing f_y in time polynomial in $\log M$. As a result Shor's algorithm gives a factor of N with high probability in time polynomial in $\log N$.

6 Grover's Algorithm

The second milestone in quantum algorithm design is Grover's algorithm for unstructured search [28, 29]. The problem of unstructured search is paradigmatic for any problem where an optimal solution needs to be found in a black box fashion, i.e., without using the possible structure of the problem:

Problem: Given a Boolean black box function $f_w : \{0,1\}^n \to \{0,1\}$ *which is equal to* 0 *for all inputs except one ("marked item"* w*), find the marked item* w*.*

Classically, a deterministic algorithm needs to make $2^n - 1$ queries to identify w in the worst case and a probabilistic algorithm still needs $O(2^n)$ queries. Grover gave a quantum algorithm that solves this problem with $O(\sqrt{2^n})$ queries and this is known to be the best possible. Grover's algorithm can hence speed up quadratically any algorithm that uses searching as a subroutine.

Grover's quantum algorithm applies the subroutine of Fig. 11 about $\sqrt{2^n}$ times. Here, the n-qubit gate $C[P]$ denotes a controlled phase; it flips the

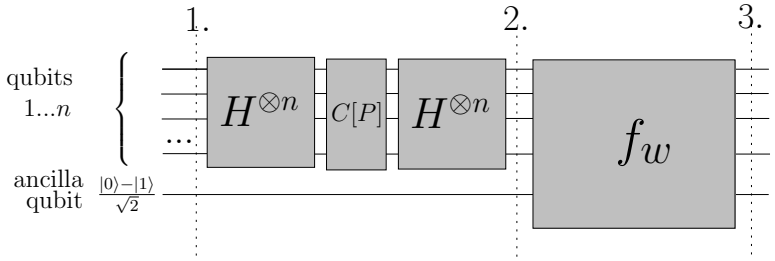

Figure 11: Subroutine in Grover's algorithm

sign of all basis states except for the all zero state. Its action can be concisely written as $C[P] = 2\,|0\ldots0\rangle\langle 0\ldots0| - I_n$, where I_n denotes the identity on n qubits. This operation is conjugated by the Hadamard transform, which maps $|0\ldots0\rangle$ to the uniform superposition $|\Psi\rangle = \frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} |x\rangle$. So the net operation between steps 1. and 2. can be written as $R_\Psi = 2\,|\Psi\rangle\langle\Psi| - I_n$. It is sometimes called diffusion or reflection around the mean, because it flips the amplitude of a state around its "mean" $\frac{1}{\sqrt{2^n}}$. The operation between steps 2. and 3. with the ancillary qubit set to $\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$ is similar to Fig. 8; it gives a phase of $(-1)^{f(x)}$ to the basis state $|x\rangle$. In our case only $f(w)$ is nonzero and so only the phase of $|w\rangle$ is flipped. This operation can be written as $R_w = I_n - 2\,|w\rangle\langle w|$. It is called reflection around w. Grover's

algorithm first applies $H^{\otimes n}$ to the state $|0\ldots0\rangle$ and then iterates T times the subroutine $R_w R_\Psi$ of Fig. 11.

Note that with input $|\Psi\rangle$ the subroutine in Fig. 11 leaves invariant the subspace spanned by $|\Psi\rangle$ and $|w\rangle$. Inside this space it acts as a real rotation with angle ϕ, where $\phi \approx \sin\phi = \frac{1}{\sqrt{2^n}}$. After T time steps the state has rotated from $|\Psi\rangle$ towards the nearly orthogonal $|w\rangle$ by an angle $T\phi$. Choosing $T = \lfloor \frac{\pi}{2}\sqrt{2^n} \rfloor$ gives a state that has overlap with $|w\rangle$ very close to 1. A measurement now gives w with very high probability.

It is not hard to see that this algorithm also works in the case of k marked items in the database; in this case its running time is $O(\sqrt{\frac{2^n}{k}})$.

7 Other Algorithms

Developments in quantum algorithm design after Shor's and Grover's algorithms can be loosely grouped into three categories: algorithms that generalize Shor's algorithm (Hidden Subgroup algorithms), algorithms that perform some version of unstructured search ("Grover-like" algorithms) and a few algorithms that don't fit into either of these categories. We mention here only a small selection of new quantum algorithms and techniques.

7.1 The Hidden Subgroup problem

Shor's algorithm can be seen as an instance of a more general problem, the Hidden Subgroup Problem. The function f in the period finding problem, viewed over $\mathbb{Z}_M$, is constant on sets $\{x, x+a, ...\}$ for each x and distinct on disjoint such sets; if a divides M it is constant on *cosets* $x + \langle a \rangle$ of the subgroup of $\mathbb{Z}_M$ generated by a and distinct on different such cosets.

Definition: The Hidden Subgroup Problem (HSP) - given a function $f : G \to R$ on a group G, and a subgroup $H < G$ such that f is constant on (left) cosets of H and distinct for different cosets, find a set of generators for H.

The HSP is an important problem. An efficient algorithm for the group $\mathbb{Z}_M$ yields an efficient factoring algorithm. It is also a component of an efficient algorithm for the discrete logarithm over $\mathbb{Z}_M$. Discrete logarithm is another cryptographic primitive in classical cryptography which would be broken by a quantum computer. Quantumly, a slight generalization of Shor's algorithm gives an efficient algorithm for HSP for all *Abelian* groups. Kitaev [36, 38, 37] developed a quantum algorithm for the Abelian Stabilizer

problem, another instance of the Hidden Subgroup Problem, using *phase estimation*, which corresponds in a way to the quantum Fourier transform and also solves the HSP over Abelian groups. Using the Abelian HSP Hallgren [30] gives a polynomial time quantum algorithm for Pell's equation, a number theoretic problem known to be at least as hard as factoring. Among other applications of the HSP, Friedl et al. [26] solve the hidden translation problem: given two functions f and g defined over some group $\mathbb{Z}_p^n$ such that $f(x) = g(x + t)$ for some hidden translation t, find t.

One of the most interesting challenges since Shor is to design quantum algorithms for the non-Abelian HSP. For instance, an efficient solution for the symmetric group S_n (permutations of n elements) would give an efficient algorithm for the Graph Isomorphism problem: to determine whether two given graphs are equal up to permutation of the vertices. Another important problem is the HSP over the dihedral group D_N (the group of symmetries of a regular N-gon). A solution in this case would give an algorithm for the shortest vector problem in a lattice; this reduction was shown by Regev [43]. The shortest vector problem is at the base of several classical cryptographic schemes designed as an alternative to those based on factoring or discrete logarithm.

In the context of the HSP over any group, Ettinger, Høyer and Knill [20] showed that a polynomial amount of coset states of the form

$$\frac{1}{\sqrt{|H|}} \sum_{h \in H} |x + h\rangle \, |f(x)\rangle$$

(compare with Eq. (3)) are enough to information theoretically obtain all the information about the hidden subgroup H. However, to *extract* this information they need exponential amount of time in the worst case; hence this algorithm is not efficient in general. For the HSP over the dihedral group D_{2^n} Kuperberg [39] gives a quantum algorithm that runs in time $2^{O(\sqrt{n})}$, a quadratic improvement in the exponent over [20] (and over any classical algorithm). There has been a lot of effort in analyzing the performance of Quantum Fourier Sampling (Fig. 9), when the QFT is the Fourier transform over the group G, when the hidden subgroup H is a subgroup of G. In the case of the symmetric group the (non-Abelian) QFT is efficiently implementable by a quantum computer [7]; however a series of papers [32, 27, 34, 40] showed that this approach to the problem cannot work (in the case of measurements of one or two and recently [31] even $\Omega(n \log n)$ copies of the state in step 4. in Fig. 9). It is an open question whether there are any efficient quantum algorithms for the HSP using other tools, not necessarily based on the QFT.

7.2 Search Algorithms

Several quantum algorithms that use Grover's search as a subroutine have been found and shown to have a polynomial speed up over their classical counterparts. For example, Brassard et al. [10] give a quantum algorithm for the problem of finding collisions in a k-to-1 function. For a k-to-1 black box function f the task is to find a collision, i.e., two inputs $x \neq y$ such that $f(x) = f(y)$. The idea is to first classically query a set K of size $|K| = (N/k)^{1/3}$ and check it for collisions, which can be done with $O((N/k)^{1/3})$ queries. If a collision is found the algorithm outputs it and stops, otherwise we set up a Grover search for a function f defined outside K that is 1 iff there is a collision with an element in K. In that case there are $(k-1)|K| \approx k^{2/3}N^{1/3}$ "marked items" and Grover's search runs in time $\sqrt{N/(k^{2/3}N^{1/3})} = (N/k)^{1/3}$. So the total number of queries of this algorithm is $O((N/k)^{1/3})$, better than any classical algorithm.

Other applications of Grover's algorithm include deciding whether all elements in the image of a function on N inputs are distinct [11], which can be done in time $O(N^{3/4})$ with Grover's algorithm as a subroutine. Note that recently a better quantum algorithm based on quantum walks has been given for this problem [4] (see next section). In [19] optimal quantum algorithms for graph problems such as (strong) connectivity, minimum spanning tree and shortest path are given using Grover's search.

7.3 Other Algorithms

Most known quantum algorithms are based on either the QFT or Grover's search. A few quantum algorithms fall outside these two frameworks. One such remarkable algorithm is for searching in an ordered list, a problem that classically takes time $\log_2 N + O(1)$. Two quantum algorithms have been given for this problem, both based on binary trees. The best known algorithm by Farhi et al. [22] finds a good quantum algorithm on a small subtree and then recurses, running with $0.526 \log_2 N$ queries. A very appealing algorithm was given by Høyer et al. [33] using the Haar-transform on the binary tree with $\log_3 N + O(1) \approx 0.631 \log_2 N + O(1)$ queries; a very interesting application of alternative efficient quantum transformations outside the QFT.

8 Recent Developments

We have seen that two types of quantum algorithms dominate the field, those that implement a version of the Hidden Subgroup problem or use the QFT and those that use a version of Grover's search. Recently, two alternative trends have entered the field, which we will briefly outline.

8.1 Quantum Walks

One of the biggest breakthroughs in classical algorithm design was the introduction of randomness and the notion of a probabilistic algorithm. Many problems have good algorithms that use a *random walk* as a subroutine. To give just one example, the currently best algorithm to solve $3SAT$ [46] is based on a random walk. With this motivation in mind, quantum analogues of random walks have been introduced. There exist two different models of a quantum walk, the continuous-time model introduced in [23] and the discrete time model of [1, 5]. The continuous model gives a unitary transformation directly on the space on which the walk takes place. The discrete model introduces an extra coin register and defines a two-step procedure consisting of a "quantum coin flip" followed by a coin-controlled walk step. The quantities important for algorithm design with random walks are their mixing time – the time it takes to be close to uniformly distributed over the domain – and the hitting time – the expected time it takes to hit a certain point. These quantities have been analyzed for several graphs in both the continuous and the discrete model. It turns out that a quantum walk can speed up the mixing time up to quadratically with respect to its classical counterpart; so the classical and quantum performance are polynomially related. The hitting behavior of a quantum walk, however, can be very different from classical. It has been shown that there are graphs and two vertices in them such that the classical hitting time from one vertex to the other is polynomial in the number of vertices of the graph, whereas the quantum walk is exponentially faster. Using this idea in [12] an (artificial) problem is constructed for which a quantum walk based algorithm gives a provable exponential speed-up over any classical probabilistic algorithm. It is open whether quantum hitting times can be used to speed up classical algorithms for relevant problems.

Based on this work a quantum walk algorithm has been introduced in [45] for the problem of finding a marked vertex in a graph. The idea is very simple: the algorithm starts in the uniform superposition over all vertices. At each step it performs a quantum walk; there are two local rules for the

walk, at an unmarked vertex the walk proceeds as usual, but at a marked vertex a different transition rule is applied (usually at an unmarked vertex a quantum coin is flipped and at a marked vertex it is not flipped). It turns out that after some time the amplitude of the state concentrates in the marked item(s); a measurement finds a marked item with high probability.

This algorithm solves Grover's problem on a graph. Why do we need a quantum walk search if we have Grover's algorithm? It turns out that there are situations when the diffusion step R_Ψ of Grover's algorithm cannot be implemented efficiently (because the local topology of the database does not allow for it, because of limitations on the quantum gates or because it is too costly in a query setting). A quantum walk only makes local transitions and can be more advantageous. One example is the search for a marked item in a 2 dimensional database. In this case Grover's algorithm requires $\sqrt{N}$ queries, but to shift amplitude form one item of the database to another on the grid takes an additional $\sqrt{N}$ steps on average per query. The net complexity of the algorithm becomes $\sqrt{N} \cdot \sqrt{N} = N$ and the quantum advantage is lost. The quantum walk algorithm has been shown to find a marked item in time $O(\sqrt{N} \log N)$ [6].

A second example of the superiority of the quantum walk search over Grover's algorithm has been given in [4]. Ambainis uses a quantum walk to give an improved algorithm for element distinctness, which runs in optimal time $O(N^{2/3})$; thus improving over Grover-based algorithms for this problem (which runs in time $O(N^{3/4})$, see Sec. 7). Several new quantum walk based algorithms with polynomial improvements over Grover-based algorithms have followed suit. For references on quantum walks see [35, 3].

8.2 Adiabatic Quantum Algorithms

Another recent alternative for algorithm design has been the introduction of adiabatic quantum algorithms by Farhi et al. [21]. The idea is the following: many optimization and constraint satisfaction problems can be encoded into a sum of local Hamiltonians $H = \sum_i H_i$ such that each term H_i represents a local constraint. The groundstate of H violates the smallest number of such constraints and represents the desired optimal solution. In order to obtain this state, another Hamiltonian H' is chosen such that the groundstate of H', $|\Phi'\rangle$, is easy to prepare. An adiabatic algorithm starts in the state $|\Phi'\rangle$ and applies H'. The Hamiltonian is then slowly changed from H' to H, usually in a linear fashion over time, such that the Hamiltonian at time t is given by $H(t) = (1 - t/T)H' + (t/T)H$. Here T is the total runtime of the algorithm. If this is done slowly enough, the adiabatic theorem guaranties

that the state at time t will be the groundstate of $H(t)$, leading to the solution, the groundstate of H, at time T. The instantaneous groundstate of the system is "tracked". But how slow is slow enough? The *adiabatic theorem* gives bounds on the speed of change of $H(t)$ such that the state stays close to the groundstate. These bounds are determined by the energies of the Hamiltonian H and by the inverse gap of the Hamiltonians $H(t)$. The gap of a Hamiltonian is the energy difference between its groundstate and first excited state, or the difference between its smallest and second smallest eigenvalue when viewed as a matrix. To design an efficient adiabatic algorithm, one has to pick H and H' such that the gap of $H(t)$ at all times t is at least inverse polynomial in the size of the problem.

Farhi et al. set up adiabatic algorithms for NP-complete problems like $3SAT$ [21]. It has been impossible so far to determine the gap analytically and the number of qubits in numerical simulations is limited. However, this approach seems promising, even though there is now mounting evidence that an adiabatic algorithm cannot solve NP-complete problems efficiently. For instance, quantum unstructured search has been implemented adiabatically and shown to have to same run-time as Grover's algorithm [44, 13].

It is not hard to see that an adiabatic algorithm can be simulated efficiently with a quantum circuit [21] – one needs to implement a time-dependent unitary that is given by a set of local Hamiltonians, each one acting only on a few qubits. Recently it has been shown [2] that also any quantum circuit can be simulated efficiently by an appropriate adiabatic algorithm; hence these two models of computation are essentially equivalent. This means that a quantum algorithm can be designed in each of the two models. The advantage of the adiabatic model is that it deals with gaps of Hermitian matrices, an area that has been widely studied both by solid state physicists and probabilists. Hopefully this new toolbox will yield new algorithms.

9 Exercises

1. *Classical Reversible Computation:* We have seen that using the 3-bit Toffoli gate one can compute any Boolean function (Toffoli gates are universal). Is there a universal set of 1- and 2-bit gates? If yes, give it, if no, prove your answer.

2. *Universality:* Give an implementation of the n-qubit gate $C[P]$ in Grover's algorithm ($C[P] = 2\left|0\ldots0\right\rangle\!\left\langle 0\ldots0\right| - I_n$ in terms of the elementary 1- and 2 qubit gates from the universal set $\{X, PI/8, H, CNOT\}$

(see Sec. 1).

3. *Bernstein-Vazirani algorithm [9]:* Give a quantum algorithm for the following problem. Given a function $f_a : \{0,1\}^n \longrightarrow \{0,1\}$, $f_a(x) = a \cdot x (= \sum_{i=1}^n a_i x_i)$ for some $a \in \{0,1\}^n$, find a with one query only. How many queries are needed in a classical deterministic algorithm? In a classical probabilistic algorithm?

4. *QFT with bounded precision:* Quantum gates cannot be implemented with perfect precision. Define the *error* of a gate U that is supposed to implement V as $E(U,V) := \max_{|v\rangle:|||v\rangle||=1} ||(U-V)|v\rangle||$. We have seen an implementation of the QFT over $\mathbb{Z}_N$ with about $\frac{1}{2}\log^2 N$ gates.
a) Show: If each gate in the QFT is implemented with error at most ϵ for some $\epsilon > 0$, then this circuit approximates the QFT with error $O(\log^2 N/\epsilon)$.
b) Give a circuit with only $O(\log N \log\log N)$ gates that for any $c > 1$ approximates the QFT to within error $1/\log^c N$.

5. *Grover with several marked items:* First, compute the run-time of Grover's algorithm when there are exactly k marked items and k is known in advance. Then, give an algorithm for Grover's problem when the number of marked items is not known.

6. *Minimum Finding [18]:* Given N distinct integers, design a quantum algorithm that finds their minimum with $O(\sqrt{N}\log N)$ queries. *Hint:* Pick a random element and use $O(\log N)$ rounds. In each round use Grover's search to replace this element with another one that is smaller.

References

[1] D. Aharonov, A. Ambainis, J. Kempe, and U. Vazirani. Quantum walks on graphs. In *Proc. 33th ACM Symp. on the Theory of Computing (STOC)*, pages 50–59, 2001.

[2] D. Aharonov, W. van Dam, J. Kempe, Z. Landau, S. Lloyd, and O. Regev. Adiabatic quantum computation is equivalent to standard quantum computation. In *Proc. 45th Annual IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 42-51, 2004.

[3] A. Ambainis. Quantum search algorithms (survey). *SIGACT News*, 35(2):22–35, 2004.

[4] A. Ambainis. Quantum walk algorithm for element distinctness. In *Proc. 45th Annual IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 22–31, 2004.

[5] A. Ambainis, E. Bach, A. Nayak, A. Vishwanath, and J. Watrous. One-dimensional quantum walks. In *Proc. 33th ACM Symp. on the Theory of Computing (STOC)*, pages 60–69, New York, NY, 2001.

[6] A. Ambainis, J. Kempe, and A. Rivosh. Coins make quantum walks faster. In *Proc. 16th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 1099–1108. 2005.

[7] R. Beals. Quantum computation of Fourier transforms over symmetric groups. In *Proc. 29th ACM Symp. on the Theory of Computing (STOC)*, pages 48–53, 1997.

[8] Ch. Bennett. Logical reversibility of computation. *IBM J. Res. Develop.*, 17:5225, 1973.

[9] E. Bernstein and U. Vazirani. Quantum complexity theory. *SIAM Journal on Computing*, 26:1411, 1997.

[10] G. Brassard, P. Hoyer, and A. Tapp. Quantum cryptanalysis of hash and claw-free functions. In *Proc. of Third Latin American Symposium on Theoretical Informatics (LATIN)*, number 1380 in LNCS, pages 163–169, 1998.

[11] H. Buhrman, C. Dürr, M. Heiligman P. Høyer, F. Magniez, M. Santha, and R. de Wolf. Quantum algorithms for element distinctness. In *Proceedings of 15th IEEE Conference on Computational Complexity*. Extended version in SIAM Journal of Computing, 34(6):1324-1330, 2005.

[12] A. M. Childs, R. Cleve, E. Deotto, E. Farhi, S. Gutmann, and D. A. Spielman. Exponential algorithmic speedup by a quantum walk. In *Proc. 35th ACM Symp. on the Theory of Computing (STOC)*, pages 59–68, 2003.

[13] W. van Dam, M. Mosca, and U. Vazirani. How powerful is adiabatic quantum computation? In *Proc. 42nd Annual IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 279–287, 2001.

[14] D. Deutsch. Quantum theory, the church-turing principle and the universal quantum computer. *Proc. Roy. Soc. London Ser. A*, 400:97–117, 1985.

[15] D. Deutsch, A. Barenco, and A. Ekert. Universality in quantum computation. *Proc. Roy. Soc. London Ser. A*, 449:669, 1995.

[16] D. Deutsch and R. Jozsa. Rapid solution of problems by quantum computation. In *Proc. Roy. Soc. London*, volume 439 of *A*, pages 553–558, 1992.

[17] D. P. DiVincenzo. Two-bit gates are universal for quantum computation. *Phys. Rev. A*, 51(2):1015–1022, 1995.

[18] C. Dürr and P. Høyer. A quantum algorithm for finding the minimum. Technical report. `http://xxx.lanl.gov/abs/quant-ph/9607014`.

[19] C. Dürr, M. Heiligman, P. Høyer, and M. Mhalla. Quantum query complexity of some graph problems. In *Proc. of 31st International Colloquium on Automata, Languages, and Programming (ICALP)*, number 3142 in LNCS, pages 481–493, 2004.

[20] M. Ettinger, P. Høyer, and E. Knill. Hidden subgroup states are almost orthogonal. *Information Processing Letters*, 91(1):43–48, 2004.

[21] E. Farhi, J. Goldstone, S. Gutmann, J. Lapan, A. Lundgren, and D. Preda. A quantum adiabatic evolution algorithm applied to random instances of an NP-complete problem. *Science*, 292(5516):472–476, 2001.

[22] E. Farhi, J. Goldstone, S. Gutmann, and M. Sipser. Invariant quantum algorithms for insertion into an ordered list. Technical report, lanl-arXive quant-ph/9901059, 1999.

[23] E. Farhi and S. Gutmann. Quantum computation and decision trees. *Phys. Rev. A*, 58:915–928, 1998.

[24] R. Feynman. Simulating physics with computers. *Internat. J. Theoret. Phys.*, 21:467–488, 1982.

[25] R. Feynman. Quantum mechanical computers. *Optics News*, 11:11–21, February 1985.

[26] K. Friedl, G. Ivanyos, F. Magniez, M. Santha, and P. Sen. Hidden translation and orbit coset in quantum computing. In *Proc. of 35th ACM Symp. on Theory of Computing (STOC)*, pages 1–9, 2003.

[27] M. Grigni, L. Schulman, M. Vazirani, and U. Vazirani. Quantum mechanical algorithms for the nonabelian hidden subgroup problem. In *Proc. 33th ACM Symp. on Theory of Computing (STOC)*, pages 68–74, 2001.

[28] L. Grover. A fast quantum mechanical algorithm for database search. In *Proc. 28th ACM Symp. on Theory of Computing (STOC)*, pages 212–219, 1996.

[29] L.K. Grover. Quantum mechanics helps in searching for a needle in a haystack. *Phys. Rev. Lett.*, 79:325, 1997.

[30] S. Hallgren. Polynomial-time quantum algorithms for pell's equation and the principal ideal problem. In *Proc. 34th ACM Symp. on Theory of Computing (STOC)*, pages 653–58, 2002.

[31] S. Hallgren, C. Moore, M. Roetteler, A. Russell, P. Sen Limits of Quantum Coset States for Graph Isomorphism. In *Proc. 38th ACM Symp. on Theory of Computing (STOC)*, pages 604–17, 2006.

[32] S. Hallgren, A. Russell, and A. Ta-Shma. Normal subgroup reconstruction and quantum computation using group representations. In *Proc. 32nd ACM Symp. on Theory of Computing (STOC)*, pages 627–635, 2000.

[33] P. Høyer, J. Neerbeck, and Y. Shi. Quantum complexities of ordered searching, sorting and element distinctness. *Algorithmica*, 34(4):429–448, 2002. Special Issue in Quantum Computation and Cryptography.

[34] J. Kempe and A. Shalev. The hidden subgroup problem and permutation group theory. In *Proc. 16th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 1118-1125, 2005.

[35] J. Kempe. Quantum random walks - an introductory overview. *Contemporary Physics*, 44(4):302–327, 2003.

[36] A. Kitaev. Quantum measurements and the abelian stabilizer problem. arXive preprint lanl quant-ph/9511026, 1995.

[37] A.Y. Kitaev, A.H. Shen, and M.N. Vyalyi. *Classical and Quantum Computation*. Number 47 in Graduate Series in Mathematics. AMS, Providence, RI, 2002.

[38] A.Yu. Kitaev. Quantum computations: Algorithms and error corrections. *Russian Math. Surveys*, 52:1191–1249, 1997.

[39] G. Kuperberg. A subexponential-time algorithm for the dihedral hidden subgroup problem. SIAM Journal of Computing, 35(1): 170-188 (2005)

[40] C. Moore, A. Russell, and L. Schulman. The symmetric group defies strong Fourier sampling. In *Proc. 46th Annual IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 479-490, 2005.

[41] M.A. Nielsen and I.L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, Cambridge, UK, 2000.

[42] J. Preskill. Quantum information and computation, Lecture notes. `http://www.theory.caltech.edu/people/preskill/ph229/`, 1998.

[43] O. Regev. Quantum computation and lattice problems. In *Proc. 43rd Annual IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 520–529, 2002.

[44] J. Roland and N. Cerf. Quantum search by local adiabatic evolution. *Phys. Rev. A*, 65:042308, 2002.

[45] N. Shenvi, J. Kempe, and K.B. Whaley. A quantum random walk search algorithm. *Phys. Rev. A*, 67(5):052307, 2003.

[46] U. Schöning. A probabilistic algorithm for k-SAT and constraint satisfaction problems. In *40th Ann. Symp. on Foundations of Computer Science*, pages 410–414. IEEE, 1999.

[47] P.W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comp.*, 26(5):1484–1509, 1997. preliminary version in *Proceedings of the 35th Ann. IEEE Symp. on the Foundations of Computer Science (FOCS)*, pages 124–134, 1994.

[48] D. Simon. On the power of quantum computation. *SIAM J. Comp.*, 26(5):1474–1483, 1997. preliminary version in *Proc. 26th ACM Symp. on Theory of Computing (STOC)*, pages 116–123, 1994.

[49] T. Toffoli. Reversible computing. In W. de Bakker and J. van Leeuwen, editors, *Automata, Languages and Programming*, page 632. Springer, New York, 1980.