8 Quantum Algorithms

8.1 Quantum versus classical algorithms

8.1.1 Why Quantum?

Quantum computers can be built as universal computers, i.e., such that they can perform all tasks that can be executed on any other (classical or quantum) computer. However, as long as they use the same algorithms for these tasks as classical computers, they also need roughly the same amount of time for completing the task. As discussed in Chapter 3, "roughly the same amount of time" refers mostly to the scaling issues, i.e., how quickly the required time increases with the size of the problem. Only when algorithms are implemented that use specific properties of quantum mechanical systems can quantum computers outperform classical computers. Such algorithms, which are known as "quantum algorithms", require hardware that is designed as a quantum computer. The Gottesman-Knill theorem [129] gives a precise description of a class of quantum algorithms that can be simulated efficiently (i.e. polynomially) on a classical computer. That class comprises all quantum computations beginning with computational-basis state preparation and involving Hadamard, phase, CNOT, and Pauli gates and measurements of observables from the Pauli group (including, for example, measurements of the computational basis).

Problems where quantum algorithms are more efficient than classical algorithms typically include many repetitions of some task on a large number of input values. A prototypical example is the search through an unstructured database, e.g., the search for a person of whom one only knows the phone number. Classical computers then have to look through all entries of the phone book in turn, comparing the listed number with the given number. As shown in the upper part of Figure 8.1, this procedure involves many repetitions of the simple task (read item - compare - decide if numbers are identical).

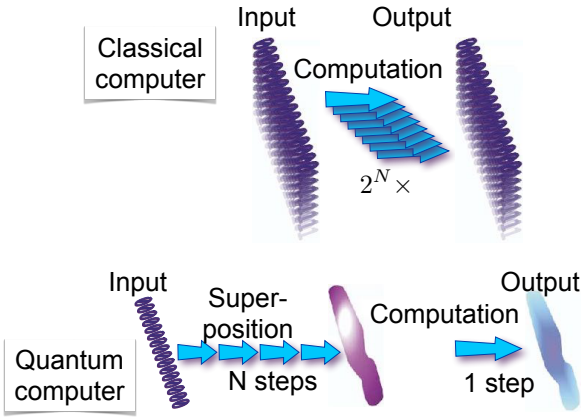

Figure 8.1: Differences between classical and quantum algorithms.

For a number of similar problems, quantum computers can search the database more efficiently. As shown in the lower part of Figure 8.1, these algorithms typically involve the following steps. The system is initially in a well defined state, which we take to be the ground state $|0\rangle$. Starting from this state, a superposition of all possible basis states is established. For a system of N qubits, the number of basis states is 2^N. The process of creating these superpositions can be completed in $O(N)$ steps; it is therefore efficient in the computational sense. The next step is the application of a transformation to this superposition state. This step is in some cases equivalent to performing the same operation on each of the 2^N states sequentially. Since this step replaces 2^N operations, it is largely responsible for the high efficiency of quantum computers compared to classical computers. This feature is often re-

ferred to as *quantum parallelism*. After this central computational step, another transformation is usually required to arrange the relevant information in the output qubits in such a way that it can be read out during the final step.

8.1.2 Classes of quantum algorithms

If we consider simple numerical tasks like multiplication for the central transformation operation, it will transform the superposition state into a superposition of the results of the multiplication. While the operation is fast, such an algorithm cannot be considered efficient, since the time for readout of the 2^N individual results would still grow exponentially with the number of qubits. The advantages of "quantum parallelism" can therefore only be exploited in cases where one is not interested in all answers to all possible inputs. Instead, quantum algorithms concentrate on two key issues: finding something (e.g., a result to a query) or determining global properties of some functions, such as the period of a function, the median of a sequence, etc., rather than individual details [130]. Accordingly, the quantum algorithms that have been introduced so far can be broadly classified into two kinds:

- Order-finding algorithms, based, e.g., on the quantum Fourier transform. The most prominent member of this class is Shor's [19] algorithm with its exponential speedup of number factoring as compared to classical algorithms.
- Quantum searching algorithms, for example the one by Grover [131, 132] with its quadratic speedup for a "needle in a haystack" search in an unstructured database.

Figure 8.2 illustrates the quantum parallelism at the example of a search algorithm. In the quantum algorithm (right), all possible paths are searched simultaneously, in parallel. It therefore finishes much faster than the classical algorithm on the left, which must follow each of the different possible paths sequentially.

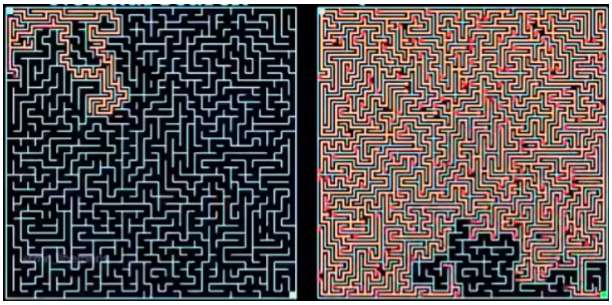

Figure 8.2: Graphical comparison between a classical (left) and a quantum search (right) algorithm.

8.2 The Deutsch algorithm: Looking at both sides of a coin at the same time

While some of the proposed algorithms involve advanced mathematical tools, others are quite easy to understand intuitively. We first discuss the relatively simple Deutsch algorithm, as an example of an order-finding algorithm. It determines global properties of certain classes of functions.

8.2.1 Functions and their properties

The Deutsch (–Jozsa) algorithm provides a possibility for computing global properties of certain functions in exponentially less time than any classical algorithm. It was originally put forward by Deutsch [16] and generalized to several input qubits by Deutsch and Jozsa [17]. The algorithm has been implemented experimentally on both ion-trap [133] and NMR quantum information processing systems [134].

While the properties of some functions are easy to describe (e.g., increasing monotonically, oscillating ...), one may also encounter functions that are too complex for such an analysis or for which no analytical expression is available. This

includes also functions that can only be called in the form of a 'black box'. In such cases, one may still be interested in finding global properties of the functions, e.g., determining if the function is constant (it's output does not depend on the input) or if it includes all possible numbers among the possible results. The Deutsch algorithm [16] and its extensions (see Section 8.2.6) provide an efficient way of answering these questions. With a single function evaluation, this algorithm distinguishes between two types of functions

$$f : x \to \{0, 1\}$$

that take positive integers as input and yield the output zero or one. The two types of functions considered are balanced (i.e., outputs zero and one occur with equal frequency) or constant (i.e., the output is either always zero or always one).

Quantum mechanically, function evaluations are implemented as unitary transformations $\mathbf{U}_f$ acting on the states that represent the information

$$\mathbf{U}_f|x\rangle = |f(x)\rangle.$$

Clearly, not every function can be represented as a unitary transformation (e.g., constant functions are manifestly non-invertible and hence non-unitary), but it is always possible to find an enlarged state space, in which a unitary operator exists that maps the possible inputs into the correct output states.

8.2.2 Example : one-qubit functions

As the simplest example, consider a one-bit-to-one-bit function $f(x)$. There are four possible one-bit-to-one-bit functions:

$$\begin{aligned} f_1 &: 0 \to 0, 1 \to 1 \\ f_2 &: 0 \to 1, 1 \to 0 \\ f_3 &: 0 \to 0, 1 \to 0 \\ f_4 &: 0 \to 1, 1 \to 1 \end{aligned}$$

which can be encoded as 2×2 matrices (compare Section 4.3):

$$\begin{aligned} f_1 = \mathbf{1} \quad , \quad f_2 = \mathbf{X} \\ f_3 = \mathbf{P}_+ + \frac{1}{\hbar}\mathbf{S}_+ \quad , \quad f_4 = \mathbf{P}_- + \frac{1}{\hbar}\mathbf{S}_-. \end{aligned}$$

The first two functions are *balanced* (both outputs 0 and 1 occur with equal frequency), the other two are constant. In the original algorithm by Deutsch [16], an additional qubit y is required to implement these functions on a quantum computer. On this quantum register (consisting of the two qubits x and y), the function evaluation is implemented as an addition without carry on the second qubit:

$$\mathbf{U}_f|x, y\rangle = |x, y \oplus f(x)\rangle$$

where $\oplus$ means addition modulo 2, or XOR.

In the actual computational basis $(|00\rangle, |01\rangle, |10\rangle, |11\rangle)$ that includes the additional bit, the first function (the one-qubit identity) corresponds to the mapping

$$\begin{aligned} U_{f_1} : |0,0\rangle &\to |0,0\rangle, \quad |0,1\rangle \to |0,1\rangle, \\ |1,0\rangle &\to |1,1\rangle, \quad |1,1\rangle \to |1,0\rangle, \end{aligned}$$

which can be written as the matrix

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

The blocks $\mathbf{1}$, $\mathbf{0}$, and $\mathbf{X}$ represent 2×2 matrices. The other three one-bit-to-one-bit functions can similarly be represented in the form

$$\begin{aligned} \mathbf{U}_{f_2} &= \begin{pmatrix} \mathbf{X} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} \end{pmatrix} \\ \mathbf{U}_{f_3} &= \begin{pmatrix} \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} \end{pmatrix} \\ \mathbf{U}_{f_4} &= \begin{pmatrix} \mathbf{X} & \mathbf{0} \\ \mathbf{0} & \mathbf{X} \end{pmatrix} \end{aligned}$$

Each of these real symmetric matrices $\mathbf{U_f}$ is its own inverse. Hence the matrices are unitary, as required for their implementation by a quantum mechanical evolution.

8.2.3 Evaluation

To compute $f(x)$ we initialize y to zero and apply $\mathbf{U_f}$ to $|x,0\rangle$:

$$\mathbf{U}_f|x,0\rangle = |x,f(x)\rangle.$$

This shows how storing the input qubit x makes constant functions invertible.

To use the advantage of quantum parallelism, we use the Hadamard gate

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

to put the first qubit into a superposition state:

$$\mathbf{H}|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle+|1\rangle); \quad \mathbf{H}|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle-|1\rangle).$$

This allows us to perform the function f on both input states simultaneously:

$$\begin{aligned}\mathbf{U}_f\mathbf{H}_x|00\rangle &= \frac{1}{\sqrt{2}}\mathbf{U}_f(|00\rangle+|10\rangle) \\ &= \frac{1}{\sqrt{2}}(|0,f(0)\rangle+|1,f(1)\rangle),\end{aligned}$$

where $\mathbf{H}_x$ means the Hadamard gate applied to the x qubit. By applying $\mathbf{U}_f$ just *once* to a superposition of two input states, we have thus obtained information about f for *both* possible input values; this is the simplest example of quantum parallelism.

8.2.4 The Deutsch algorithm

The original Deutsch algorithm uses a 2-qubit system that is initially in the state $|\psi_0\rangle = |0,1\rangle$. Applying Hadamard gates to both qubits brings it into the superposition state

$$\begin{aligned}|\psi_1\rangle &= \mathbf{H}_x\mathbf{H}_y|0,1\rangle = \frac{1}{2}\left(|0\rangle+|1\rangle\right)\left(|0\rangle-|1\rangle\right) \\ &= \frac{1}{2}\left(|00\rangle+|10\rangle-|01\rangle-|11\rangle\right).\end{aligned}$$

Applying the function operator $\mathbf{U}_f$ to this state yields

$$\begin{aligned}|\psi_2\rangle &= \mathbf{U}_f|\psi_1\rangle \\ &= \frac{1}{2}(|0,f(0)\rangle+|1,f(1)\rangle \\ &\quad -|0,1\oplus f(0)\rangle-|1,1\oplus f(1)\rangle).\end{aligned}$$

As is often the case in quantum algorithms, the input values are now entangled with the function results.

We now distinguish the two cases where the function is either constant ($f(0)=f(1)$) or balanced ($f(1)=1\oplus f(0)\neq f(0)$). In the first case the quantum register is in the state

$$\begin{aligned}|\psi_2\rangle &= \frac{1}{2}\left(|0,f(0)\rangle+|1,f(0)\rangle-|0,1\oplus f(0)\rangle\right. \\ &\quad \left.-|1,1\oplus f(0)\rangle\right) \\ &= \frac{1}{2}\left(|0\rangle+|1\rangle\right)\left(|f(0)\rangle-|1\oplus f(0)\rangle\right).\end{aligned}$$

In the second (balanced) case, we use $f(1) = 1\oplus f(0)$ and the state is

$$\begin{aligned}|\psi_2\rangle &= \frac{1}{2}\left(|0,f(0)\rangle+|1,1\oplus f(0)\rangle\right. \\ &\quad \left.-|0,1\oplus f(0)\rangle-|1,f(0)\rangle\right) \\ &= \frac{1}{2}\left(|0\rangle-|1\rangle\right)\left(|f(0)\rangle-|1\oplus f(0)\rangle\right).\end{aligned}$$

Comparing these two states we see that the answer to our question (function constant or balanced) is now encoded in the relative phase of the x qubit. A measurement of $\mathbf{X}$ will therefore reveal the answer.

Alternatively, the information can be converted into the populations of that qubit by a second application of the Hadamard gate:

$$\begin{aligned}|\psi_3\rangle &= \mathbf{H}_x|\psi_2\rangle \\ &= |f(0)\oplus f(1)\rangle\left(\frac{|f(0)\rangle-|1\oplus f(0)\rangle}{\sqrt{2}}\right).\end{aligned}$$

The x qubit contains now the sum (modulo 2) of the two possible function values. It is therefore zero if they are equal, i.e., the function

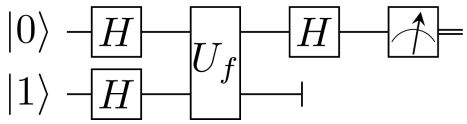

Figure 8.3: Summary of the Deutsch algorithm.

is constant, and 1 if the function is balanced ($f(0) \oplus f(1) = 1$). Figure 8.3 summarizes the algorithm.

One function evaluation is thus enough to determine whether f is balanced or constant. A pictorial way of describing this is "looking at both sides of a coin at the same time": if the two sides of a coin are equal, it is forged (not too cleverly, however), if not, chances are that it is good.

8.2.5 Many qubits

The one-qubit Deutsch algorithm is not too impressive, but consider now a function with n input qubits, and still only one output qubit. Now, we try to answer the question if a function

$$f(x_1, \ldots, x_n) = \begin{cases} 0 \\ 1 \end{cases}$$

is constant or balanced.

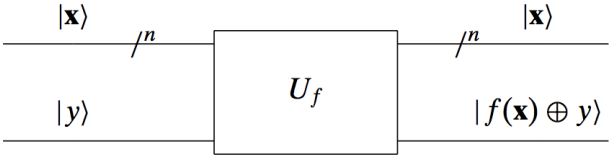

Figure 8.4: Function evaluation in the Deutsch-Jozsa algorithm.

The initial state of the quantum register is now

$$|\psi_0\rangle = |\vec{0}, 1\rangle = |0\rangle_1 |0\rangle_2 \ldots |0\rangle_{n-1} |0\rangle_n |1\rangle_{n+1}.$$

Applying the $n+1$-qubit Hadamard transformation

$$\mathbf{H}_{\vec{x}} \mathbf{H}_y = \prod_{i=1}^{n+1} \mathbf{H}_i = \mathbf{H}_1 \otimes \mathbf{H}_2 \otimes \ldots \mathbf{H}_n \otimes \mathbf{H}_{n+1}$$

(with $\mathbf{H}_i$ the Hadamard gate acting on qubit i) to this state yields

$$\begin{aligned} |\psi_1\rangle &= \mathbf{H}_{\vec{x}} \mathbf{H}_y |\vec{0}, 1\rangle \\ &= \left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right)_1 \left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right)_2 \\ &\quad \ldots \left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right)_n \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right)_y \\ &= \frac{1}{\sqrt{2^{(n+1)}}} \sum_{\vec{x}} |\vec{x}\rangle \left(|0\rangle - |1\rangle\right)_y, \end{aligned}$$

a superposition of all possible input states. The sum runs over all possible n-bit vectors $\vec{x}$. This step is extremely efficient: it takes only $n+1$ operations (which often can be performed in parallel) to create an equal-weight superposition of the 2^{n+1} input states.

The functions to be examined are again implemented by the unitary operation

$$\mathbf{U}_f |\vec{x}, y\rangle = |\vec{x}, y \oplus f(\vec{x})\rangle.$$

Applying this transformation to the superposition of all input states yields

$$|\psi_2\rangle = \mathbf{U}_f |\psi_1\rangle.$$

Using

$$\begin{aligned} & \mathbf{U}_f(|\vec{x}\rangle(|0\rangle - |1\rangle)) \\ = \; & |\vec{x}\rangle(|f(\vec{x})\rangle - |1 \oplus f(\vec{x})\rangle) \\ = \; & \begin{cases} |\vec{x}\rangle(|0\rangle - |1\rangle) & \text{for } f(\vec{x}) = 0 \\ |\vec{x}\rangle(|1\rangle - |0\rangle) & \text{for } f(\vec{x}) = 1 \end{cases} \end{aligned}$$

we find

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

The possible function values are now stored in the signs of the amplitudes in the superposition state.

The final step of the algorithm is another Hadamard transformation, as in the one-qubit

case. To understand its effect, consider a Hadamard gate applied to a single qubit $|x\rangle$:

$$\begin{aligned}\mathbf{H}|x\rangle &= \frac{1}{\sqrt{2}}(|0\rangle + (-1)^x|1\rangle) \\ &= \frac{1}{\sqrt{2}}\sum_z (-1)^{xz}|z\rangle.\end{aligned}$$

Here, z represents the possible values of the output qubit.

This generalizes to the n-qubit case:

$$\mathbf{H}_{\vec{x}}|\vec{x}\rangle = \frac{1}{\sqrt{2^n}}\sum_{\vec{z}}(-1)^{\vec{x}\cdot\vec{z}}|\vec{z}\rangle$$

where $\vec{x}\cdot\vec{z} = \sum_i x_i z_i$ is the bitwise scalar product of the two n-qubit vectors $\vec{x}$ and $\vec{z}$. The final state of the n-qubit algorithm is therefore

$$\begin{aligned}|\psi_3\rangle &= \mathbf{H}_{\vec{x}}|\psi_2\rangle \\ &= \frac{1}{2^n}\sum_{\vec{z}}\sum_{\vec{x}}(-1)^{\vec{x}\cdot\vec{z}+f(\vec{x})}|\vec{z}\rangle\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right).\end{aligned}$$

To decide if the function is constant or balanced, one has to measure the population of the ground state $|\vec{z}\rangle = |\vec{0}\rangle$, whose amplitude is

$$2^{-n}\sum_{\vec{x}}(-1)^{f(\vec{x})} = \begin{cases} \pm 1 \text{ for } f \text{ constant} \\ 0 \text{ for } f \text{ balanced} \end{cases},$$

and obviously some intermediate value if f is neither balanced nor constant.

8.2.6 Extensions and generalizations

The Deutsch–Jozsa algorithm performs the test (balanced or constant) on a n-bit function $f(\vec{x})$. If one imagines that n may be large and f may be costly to evaluate, then the advantage of having only one function evaluation (as compared to $\mathcal{O}(2^n)$) is clear. It is, however, important to stress that the function must be *promised* to be either balanced or constant; for a more general function the Deutsch–Jozsa algorithm will give an ambiguous answer.

The algorithm was improved in Ref. [135] and generalized to mixed (thermal) states in Ref. [134]. An interesting generalization was published by Chi, Kim and Lee [136]: they showed that the scheme can be extended to functions whose results are integers rather than bits. Furthermore, their modification does not require the auxiliary qubit $|y\rangle$, which is modified in the Deutsch–Jozsa algorithm, but whose state is not needed for readout.

All these algorithms do not have a great practical value as compared to the Shor and Grover algorithms but they are easy to understand and they illustrate how interference, and in a way, the Fourier transform (which is related to the Hadamard transform), are employed in quantum information processing. Another Fourier-based algorithm which is more difficult, and potentially much more interesting, is Shor's algorithm for finding prime factors.

8.3 Shor's factorization algorithm

8.3.1 Overview

Shor's algorithm draws from two main sources. One source is number theory, which we will not treat too deeply, and which shows that factoring can be reduced to finding the period of certain functions. Finding a period is of course related to the physicist's everyday business of Fourier transformation, which is the second source of Shor's algorithm. A quantum computer can very effectively compute the desired number-theoretic function for many input values in parallel. The quantum version of the Fast Fourier Transform (FFT) can perform certain aspects of the Fourier transform so efficiently that the term "quantum Fourier transformation" (QFT) has been coined.

Why is it interesting to find prime factors of large numbers? The scientist's motivation is, because it is a hard problem. It turns out that this is one of the extremely rare cases where the same

motivation is shared by scientists, bankers, and the military. The reason is cryptography, the secret transmission of (for example financial or military) data by so-called public key cryptographic schemes. In these schemes a large number (the public key of the recipient) is used to generate a coded message which is then sent to a recipient. The message can only be decoded using the prime factors of the public key. These prime factors (the private key) are only known to the recipient (bank, chief of staff,...). An extremely low-level example is the number 29083=127·229. Using only pencil and paper, it will probably take you some time to find the prime factors, whereas the inverse operation (the multiplication) should not take you more than about a minute. In the present section we discuss Shor's algorithm theoretically. The experimental implementation by means of liquid-state NMR will be treated in Section 10.3.

In general, public key cryptographic systems rely on functions

$$\begin{aligned} f_a(x_i) \quad = \quad & y_i; \quad x_i \in \text{message text}, \\ & y_i \in \text{encrypted text}, \\ & a \,\text{integer, the key.} \end{aligned}$$

The inverse function

$$\tilde{f}_{(p,q)}(y_i) = x_i, \quad pq = a$$

depends on the prime factors p and q of a, which are not easy to find. This makes code breaking expensive: the number of computational steps grows as

$$\#\,\text{steps} \;\propto\; e^{cN^{1/3}(\log N)^{2/3}}.$$

In contrast, the number of steps in the quantum algorithm of Peter Shor is polynomial:

$$\#\,\text{steps} \;\propto\; N^2(\log N)(\log\log N).$$

8.3.2 Some number theory

Finding a gcd

Let $N \geq 3$ be the large odd integer which we want to factorize, and $a < N$ some other integer. Let us assume that the greatest common divisor[1] $\gcd(N, a) = 1$, that is, N and a are coprime. (If they are not coprime, $f = \gcd(N, a)$ is already a nontrivial prime factor of N and we restart with N/f in place of N.)

To determine the gcd we can employ a nice little piece of classical Greek culture, Euclid's algorithm, which is, by modern terms, an efficient algorithm. It works as follows: Let x and y be two integers, $x > y$, and $z = \gcd(x, y)$. Then both x and y as well as the numbers $x-y$, $x-2y$, ... are multiples of z, and so is the *remainder* $r = x - ky < y$ obtained in the integer division of x by y. If r is zero, $z = y$ and the problem is solved. If $r \neq 0$, the problem is transformed to a similar one involving smaller numbers:

$$z = \gcd(x, y) = \gcd(y, r). \tag{8.1}$$

The above argument can be repeated with the pair of numbers (y, r) in place of (x, y), etc. Thus z is expressed as the gcd of pairs of ever smaller numbers. The last nonzero remainder obtained in this procedure is the desired number z.

Modular exponentiation

To proceed in our attempt at factorizing the number N we need another building block from number theory, which is *modular exponentiation*. Remembering that a and N are coprime, we consider the powers a^x of a, modulo N (that is, we calculate the remainder of a^x with respect to division by N). The smallest positive integer r such that

$$a^r \bmod N = 1$$

is called the *order* of a mod N. This means that

$$a^r = k \cdot N + 1$$

for some k, and consequently

$$a^{r+1} = k \cdot N \cdot a + a$$

[1] the largest positive integer that divides each of the integers

such that

$$a^{r+1} \bmod N = a \bmod N$$

which shows that r is the *period* of the modular exponential function

$$F_N(x) = a^x \bmod N. \tag{8.2}$$

Incidentally, this shows that $r \leq N$ because $F_N(x)$ (being the remainder of a division by N) cannot assume more than N different values before repeating.

Three cases may arise:

1) r is odd,

2) r is even and $a^{r/2} \bmod N = -1$,

3) r is even and $a^{r/2} \bmod N \neq -1$.

Cases 1) and 2) are irrelevant for the factorization of N, but in case 3) at least one of the two numbers $\gcd(N, a^{r/2} \pm 1)$ is a nontrivial factor of N, as we shall show below.

8.3.3 Factoring strategy

We now show that case 3) above leads to a nontrivial factor of N. For ease of notation let us call $a^{r/2} = x$. From $x^2 \bmod N = 1$ it follows that $x^2 - 1 = (x+1)(x-1)$ is divided by N and thus N must have a common factor with $x+1$ or $x-1$. That common factor cannot be N itself, since $x \bmod N \neq -1$ and thus $x+1$ is not a multiple of N; neither can $x-1$ be a multiple of N since if it were, $a^{r/2} \bmod N = 1$ and the order would be $r/2$, not r. (Remember that the order was defined as the *smallest* number such that $a^r \bmod N = 1$.) The common factor we are looking for must then be one of the numbers $\gcd(N, a^{r/2} \pm 1)$, and the gcd can be efficiently computed by Euclid's algorithm.

Next we must make sure that case 3) above has a fair chance to occur if we randomly try some numbers a. The following facts give us hope:

- If N is a pure prime power $N = p^s$ $(s \geq 2)$, this can be detected efficiently, because then the condition $s = \frac{\log N}{\log p}$ (with integer p) must hold, which can be checked for all possible values of s. (s can be at most $\frac{\log N}{\log 2}$)
- If N is an odd composite number $N = p_1^{\alpha_1} \cdots p_m^{\alpha_m}$ $(m \geq 2)$ and a a randomly chosen integer $1 \leq a \leq N-1$ coprime to N, and $a^r = 1 \bmod N$ (that is, r is the order of $a \bmod N$), then the probability

$$\begin{aligned} & \text{prob}(r \text{ even and } a^{r/2} \bmod N \neq -1) \\ \geq \;& 1 - \frac{1}{2^m} \geq \frac{3}{4}. \end{aligned}$$

This means that for each time we calculate the order of $a \bmod N$ we have a chance of better than 75% to find a nontrivial prime factor of N. Computing the order m times reduces the chance of failure to 4^{-m}. The chance of finding a prime factor (if one exists!) can thus be brought arbitrarily close to 1, but it is important to note that Shor's is a *probabilistic* algorithm.

The proof of this number-theoretic result can be found in [40], Appendix 4. It is not difficult, but it involves a few more pieces of classical culture, such as the Chinese Remainder Theorem, which is more than 750 years old. The proof can also be found in Appendix B of the excellent 1996 paper [137] by Ekert and Jozsa.

We are now able to give an algorithm which (with high probability) returns a non-trivial factor of any composite N. All steps can be performed efficiently on a classical computer, except for the task of computing the order, which is where quantum computing comes in.

1) If N is even, return the factor 2.

2) Determine whether $N = p^s$ for integers $p \geq 3$ and $s \geq 2$, and if so return the factor p.

3) Randomly choose a in the range 1 to $N-1$. If $\gcd(a, N) > 1$ then return the factor $\gcd(a, N)$.

4) Use the order-finding subroutine to find the order of a modulo N.

5) If r is even and $a^{r/2} \bmod N \neq N-1$ then compute $\gcd(a^{r/2} \pm 1, N)$ and test to see

which one of these is a non-trivial factor, returning that factor if so. Otherwise, the algorithm fails in which case one must restart at step 3).

In Section VI of [137] the authors discuss the complete application of the algorithm to the smallest odd composite number which is not a power of a prime, $N = 15$. That number was also factorized in the first liquid-state NMR implementation of Shor's algorithm, compare Section 10.3.

8.3.4 The core of Shor's algorithm

The centerpiece of Shor's algorithm is the calculation of the order r of a mod N, that is, the period of the modular exponential function (8.2). The strategy for doing this is to calculate the function $F_N(x)$ for many values of x in parallel and to use Fourier techniques to detect the period in the sequence of function values. For a given N, two quantum registers are needed:

- a source register with K qubits such that $N^2 \leq Q := 2^K \leq 2N^2$ and
- a target register with N or more basis states, that is, at least $\log_2 N$ qubits.

Step 1 of the algorithm is the initialization of both registers

$$|\psi_1\rangle = |\vec{0}\rangle|\vec{0}\rangle.$$

1^{st} Quantum-Fouriertransform

Step 2 is the "Quantum Fourier transformation" of the source register. The quantum Fourier transformation is nothing but the ordinary discrete Fourier transformation of a set of data of length Q (details will be discussed in section 8.3.6). The corresponding unitary operator on the source register Hilbert space is defined by

$$\mathbf{U}_{F_Q} : |q\rangle \mapsto \frac{1}{\sqrt{Q}} \sum_{q'=0}^{Q-1} \exp\left(2\pi i \frac{q'q}{Q}\right) |q'\rangle.$$

The number q between 0 and $Q-1$ has the binary expansion $q = \sum_{j=0}^{K-1} q_j 2^j$, and $|q\rangle$ is shorthand for $|q_{K-1} \ldots q_1 q_0\rangle$. The target register is not modified, so the state after step 2 is

$$|\psi_2\rangle = (\mathbf{U}_{F_Q} \otimes \mathbf{1})|\psi_1\rangle = Q^{-1/2} \sum_{q=0}^{Q-1} |q\rangle|\vec{0}\rangle;$$

all the Fourier phase factors are equal to unity since all source qubits were initially zero. This particular output can also be generated by a Hadamard transform of the source register.

Modular Exponentiation

Step 3 is the application of the gate $\mathbf{U}_a$, which implements the modular exponentiation $q \mapsto f(q) = a^q \bmod N$ (we will not discuss in detail how to build this gate). The result is

$$|\psi_3\rangle = \mathbf{U}_a|\psi_2\rangle = Q^{-1/2} \sum_{q=0}^{Q-1} |q\rangle|a^q \bmod N\rangle.$$

Here $Q > N^2$ function values of the function $F_N(q)$ are computed in parallel in one step, and since $r < N$ the period r must show up somewhere in this sequence of function values. The implementation of the modular exponential is an efficient computation, using powers x_p of some integer x. First generate the $M+1$ numbers $x, x^2, x^4, ..., x^{2^M}$ by M integer multiplications. Using the binary expansion of p, x^p then can be computed using only of the order of $\log_2 p$ multiplications.

2^{nd} Quantum-Fouriertransform

Step 4: Apply the quantum Fourier transform again to the source register. This leads to

$$\begin{aligned} |\psi_4\rangle &= (\mathbf{U}_{\mathbf{F_Q}} \otimes \mathbf{1})|\psi_3\rangle \\ &= Q^{-1} \sum_{q=0}^{Q-1} \sum_{q'=0}^{Q-1} e^{2\pi i \frac{qq'}{Q}} |q\rangle|a^{q'} \bmod N\rangle \end{aligned}$$

where $f(q)$ is the function whose periodicity we are looking for.

Measure Source Qubits

Step 5: Measure the source qubits in the computational basis. The probability of finding the source register in the state q displays a pattern (due to quantum interference) from the regularities of which the order r can be deduced. To see how this comes about we assume[2] for the moment that Q is divisible by r, that is,

$$Q = nr. \tag{8.3}$$

We introduce a shorthand notation for the state $|\psi_4\rangle$:

$$|\psi_4\rangle = \sum_q \sum_{q'} \alpha_{qq'} |q\rangle |f(q')\rangle, \tag{8.4}$$

where both sums extend from 0 to $Q-1$. The probability of finding the source register in a particular basis state $|q_0\rangle$ is the expectation value of $\mathbf{P}_{q_0} \otimes \mathbf{1}$ where $\mathbf{P}_{q_0} = |q_0\rangle\langle q_0|$ is the projection operator onto $|q_0\rangle$ and the unit operator $\mathbf{1}$ acts on the target qubit:

$$\begin{aligned} & \langle\psi_4|\mathbf{P}_{q_0} \otimes \mathbf{1}|\psi_4\rangle \\ = & \sum_p \sum_{p'} \sum_q \sum_{q'} \alpha^*_{pp'} \alpha_{qq'} \langle p|q_0\rangle\langle q_0|q\rangle \\ & \langle f(p')|f(q')\rangle \qquad (8.5) \\ = & \sum_{p'} \sum_{q'} \alpha^*_{q_0 p'} \alpha_{q_0 q'} \langle f(p')|f(q')\rangle. \qquad (8.6) \end{aligned}$$

The modular exponential function $f(p) = a^p \bmod N$ has period r, and the r function values within a period are all distinct due to the nature of the function. The scalar product $\langle f(p')|f(q')\rangle$ of the target register states thus is periodic in both variables p' and q' and we can sort the terms in (8.6) according to the nonzero values of $\langle f(p')|f(q')\rangle$.

We first consider the case $p' = 0$. The scalar product $\langle f(0)|f(q')\rangle = \langle f(0)|f(0)\rangle = 1$ for $q' = 0, r, 2r, \ldots, (n-1)r$. For any of these q' values $\langle f(p')|f(0)\rangle = 1$ for $p' = 0, r, 2r, \ldots, (n-1)r$. The terms in (8.6) containing the nonzero scalar product $\langle f(0)|f(0)\rangle$ thus generate the following contribution:

$$\sum_{\nu=0}^{n-1} \sum_{\mu=0}^{n-1} \alpha^*_{q_0,\nu r} \alpha_{q_0,\mu r} = \left| \sum_{\mu=0}^{n-1} \alpha_{q_0,\mu r} \right|^2 . \tag{8.7}$$

In a similar way we can collect the contributions associated with $\langle f(1)|f(1)\rangle$, $\langle f(2)|f(2)\rangle$, ..., $\langle f(r-1)|f(r-1)\rangle$ to obtain the desired probability of finding the source register in the basis state $|q_0\rangle$:

$$\langle\psi_4|\mathbf{P}_{q_0} \otimes \mathbf{1}|\psi_4\rangle = \sum_{j=0}^{r-1} \left| \sum_{\mu=0}^{n-1} \alpha_{q_0,\mu r+j} \right|^2 . \tag{8.8}$$

The inner summation always comprises n terms, independent of j. This is due to the simplifying assumption (8.3). Without that assumption, that is, for $(n-1)r < Q < nr$ the inner sum would only have $n-1$ terms for some j. Given that we are typically discussing large numbers this is not a big effect. Re-expanding the abbreviation $\alpha_{qq'}$ introduced above, we obtain

$$\begin{aligned} & \left| \sum_{\mu=0}^{n-1} \alpha_{q_0,\mu r+j} \right|^2 \\ = & \frac{1}{Q^2} \left| \sum_{\mu=0}^{n-1} \exp\left(2\pi i \frac{q_0}{Q} (\mu r + j) \right) \right|^2 \\ = & \frac{1}{Q^2} \left| \exp\left(2\pi i \frac{q_0 j}{Q} \right) \sum_{\mu=0}^{n-1} \left(\exp\left(2\pi i \frac{q_0 r}{Q} \right) \right)^\mu \right|^2 . \end{aligned}$$

The phase factor in front of the sum is irrelevant. The (geometric) sum itself yields n if $\frac{q_0 r}{Q}$ is integer, and zero otherwise, independent of j. The probability (8.8) thus shows a regular pattern of peaks of equal height from which r may be deduced.

Without the simplifying assumption (8.3) the pattern is not quite as regular, but the probability for finding the source register in the state

[2] Although this assumption is strictly impossible since Q is a power of two it does not have major harmful effects, as we will see below.

$|q_0\rangle$ can still be expressed in terms of a few geometric sums:

$$\begin{aligned} &\langle\psi_4|\mathbf{P}_{q_0}\otimes\mathbf{1}|\psi_4\rangle \qquad (8.9)\\ &= \frac{1}{Q^2}\sum_{j=0}^{r-1}\left|\sum_{\mu=0}^{\mathrm{int}\left(\frac{Q-1-j}{r}\right)}\left(\exp\left(2\pi i\frac{q_0 r}{Q}\right)\right)^{\mu}\right|^2, \end{aligned}$$

where "int" denotes the integer part of a real number. The function (8.9) is shown in Figure 8.5 for $Q = 256$ and $r = 10$. From the regularities of peak structures like the one in Figure 8.5 the order r can be deduced with a high probability (but not with certainty) if the positions of a sufficiently large number of peaks are taken into account. We do not reproduce the technical details here and instead refer our readers to the literature for the full discussion which requires some additional mathematical tools.

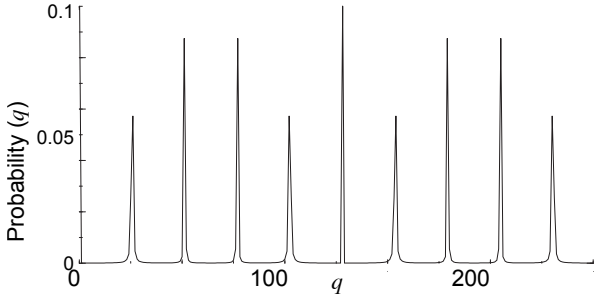

Figure 8.5: Probability of measuring q, with $Q = 256$ and $r = 10$.

Implementation of the Modular Exponentiation

What remains to be understood is the implementation of modular exponentiation and of the discrete Fourier transform. We skip all details of the modular exponentiation except for one remark related to the efficient computation of (high) powers a^x of some integer a. By M integer multiplications the $M+1$ numbers $a, a^2, a^4, \ldots, a^{2^M}$ can be generated. Given the binary expansion $x = \sum_{i=0}^{M} x_i 2^i$ ($x_i = 0, 1$) of x, the desired power can be written as

$$a^x = \prod_{i=0}^{M}\left(a^{2^i}\right)^{x_i}. \qquad (8.10)$$

Since this product contains at most $M+1$ factors the large power a^x thus can be computed using only of the order of $\log_2 x$ multiplications. The only other ingredient needed is an algorithm for multiplying two integers by means of quantum gates, which is available.

8.3.5 The Classical Fourier transform

We will first discuss the "classical" discrete Fourier transform, with a short digression on the fast Fourier transform (FFT) and then we will turn to the quantum Fourier transform (QFT) and see that it is even faster than the fast Fourier transform.

Discrete Fourier transform

The usual discrete Fourier transform maps a complex input vector with components $x_0, x_1, \ldots, x_{N-1}$ to the output vector (the Fourier coefficients) $y_0, y_1, \ldots, y_{N-1}$ by means of

$$y_k = N^{-\frac{1}{2}}\sum_{j=0}^{N-1}\exp\left(\frac{2\pi i}{N}kj\right)x_j, \qquad (8.11)$$

and vice versa,

$$x_k = N^{-\frac{1}{2}}\sum_{j=0}^{N-1}\exp\left(-\frac{2\pi i}{N}kj\right)y_j. \qquad (8.12)$$

Note that both transformations can be interpreted as "matrix times vector" operations. That the two matrices involved are in fact inverses of each other, follows from the identity

$$\sum_{k=0}^{N-1}\exp\left(\frac{2\pi i}{N}(j-l)k\right) = N\delta_{jl}, \qquad (8.13)$$

which is a geometric sum. Obviously the evaluation of the Fourier transform involves roughly

N^2 complex multiplications, and about the same number of additions. Doubling the size of the data set thus means quadrupling the operation count.

Fast Fourier transform

The FFT (which can be traced back to work by Gauß in 1805 [138]) rests on the observation that by separating even and odd j in (8.11) one obtains

$$\begin{aligned} y_k &= N^{-\frac{1}{2}} \left[\sum_{l=0}^{\frac{N}{2}-1} \exp\left(\frac{2\pi i}{N/2} kl\right) x_{2l} \right. \\ &\quad + \exp\left(\frac{2\pi i}{N} k\right) \\ &\quad \left. \cdot \sum_{l=0}^{\frac{N}{2}-1} \exp\left(\frac{2\pi i}{N/2} kl\right) x_{2l+1} \right] \end{aligned} \tag{8.14}$$

where N was assumed to be even. Note that the two sums are both again discrete Fourier transforms of $\frac{N}{2}$ data each, leading to an operation count of $2\left(\frac{N}{2}\right)^2 = \frac{1}{2}N^2$. The operation count thus has been cut in half by a simple reorganization of the Fourier sum, and there is no reason to stop at this point if $\frac{N}{2}$ is even. Continuation of this process for $N = 2^n$ yields the FFT algorithm (see, for example, [139] for details) which reduces the operation count from $\mathcal{O}(N^2)$ to $\mathcal{O}(N \log N)$ which for many applications, for example in image processing, computerized tomography, etc., makes the difference between "possible in principle only" and "practical and convenient for everyday use".

8.3.6 The quantum Fourier transform

The quantum Fourier transform is an operator defined by the following mapping of the basis states of an N-dimensional Hilbert space:

$$|j\rangle \mapsto N^{-\frac{1}{2}} \sum_{k=0}^{N-1} \exp\left(\frac{2\pi i}{N} jk\right) |k\rangle. \tag{8.15}$$

An arbitrary quantum state with amplitudes x_j is then mapped as

$$\sum_{j=0}^{N-1} x_j |j\rangle \mapsto \sum_{k=0}^{N-1} y_k |k\rangle \tag{8.16}$$

with y_k given by the "classical" Fourier transform formula (8.11). This transformation is unitary, so it conserves the norm of a quantum state:

$$\begin{aligned} \sum_{k=0}^{N-1} |y_k|^2 &= N^{-1} \sum_{k=0}^{N-1} \left| \sum_{j=0}^{N-1} x_j \exp\left(\frac{2\pi i}{N} jk\right) \right|^2 \\ &= N^{-1} \sum_{k=0}^{N-1} \sum_{j=0}^{N-1} \sum_{l=0}^{N-1} x_j x_l^* e^{\frac{2\pi i}{N}(j-l)k} \\ &= \sum_{l=0}^{N-1} |x_l|^2, \end{aligned}$$

where in the last step we have used the identity (8.13).

Let us now assume that $N = 2^n$ such that the basis states $\{|0\rangle \dots |2^n - 1\rangle\}$ form the computational basis for a n-qubit quantum computer. We will denote these basis states either by the integer j, or by the sequence $j_1 j_2 \dots j_n$ from the binary representation of j

$$j = j_1 2^{n-1} + \dots + j_n 2^0 = \sum_{\nu=1}^{n} j_\nu 2^{n-\nu}.$$

We will also need the binary representation of a fractional number (between 0 and 1) which we write as a *binary fraction*

$$\begin{aligned} & 0\,.\, j_l j_{l+1} \dots j_m \\ = \; & j_l 2^{-1} + j_{l+1} 2^{-2} + \dots + j_m 2^{-m+l-1}. \end{aligned}$$

We take another look at the quantum Fourier transform

$$|j\rangle \mapsto 2^{-\frac{n}{2}} \sum_{k=0}^{2^n-1} \exp\left(\frac{2\pi i}{2^n} jk\right) |k\rangle,$$

and insert the binary expansion of k, which leads to

$$\begin{aligned}
&|j\rangle \mapsto 2^{-\frac{n}{2}} \sum_{k_1=0}^{1} \cdots \sum_{k_n=0}^{1} \\
&\exp\left(\frac{2\pi i}{2^n} j \left(\sum_{l=1}^{n} k_l 2^{n-l}\right)\right) |k_1 \ldots k_n\rangle \\
= \; & 2^{-\frac{n}{2}} \sum_{k_1=0}^{1} \cdots \sum_{k_n=0}^{1} \bigotimes_{l=1}^{n} \exp(2\pi i j k_l 2^{-l}) |k_l\rangle \\
= \; & 2^{-\frac{n}{2}} \bigotimes_{l=1}^{n} \left[\sum_{k_l=0}^{1} \exp(2\pi i j k_l 2^{-l}) |k_l\rangle\right] \\
= \; & 2^{-\frac{n}{2}} \bigotimes_{l=1}^{n} \left[|0\rangle_l + \exp(2\pi i j 2^{-l}) |1\rangle_l\right].
\end{aligned}$$

In the first step $|k_1 \ldots k_n\rangle$ has been decomposed into an explicit tensor product $\bigotimes_{l=1}^{n} |k_l\rangle$, and in the following step sums have been rearranged according to the familiar pattern $\sum_i \sum_j a_i b_j = (\sum_i a_i)(\sum_j b_j)$. A closer look at the exponent reveals a binary fraction

$$\begin{aligned}
j2^{-l} &= \sum_{\nu=1}^{n} j_\nu 2^{n-\nu-l} \\
&= j_1 j_2 \ldots j_{n-l} \cdot j_{n-l+1} \ldots j_n.
\end{aligned}$$

The integer part (left of the decimal point) is irrelevant because $e^{i2\pi k} = 1$ and we can write the quantum Fourier transform as

$$\begin{aligned}
|j\rangle \;\mapsto\; & 2^{-\frac{n}{2}} \left(|0\rangle_1 + e^{i2\pi\, 0\,\cdot\, j_n} |1\rangle_1\right) \\
& \cdots \left(|0\rangle_2 + e^{i2\pi\, 0\,\cdot\, j_{n-1} j_n} |1\rangle_2\right) \\
& \cdots \left(|0\rangle_n + e^{i2\pi\, 0\,\cdot\, j_1 j_2 \ldots j_n} |1\rangle_n\right).
\end{aligned}$$

8.3.7 Gates for the QFT

The quantum Fourier transform is thus nothing but a simple qubit-wise phase shift: the $|1\rangle$ state of each of the n qubits is given an extra phase factor. That operation can be performed efficiently by a quantum circuit combining some simple quantum gates. Let us define the unitary (phase shift) operator

$$\mathbf{R}_k = \begin{pmatrix} 1 & 0 \\ 0 & e^{2\pi i 2^{-k}} \end{pmatrix}$$

and the corresponding controlled-$\mathbf{R}_k$ gate which applies $\mathbf{R}_k$ to the target qubit if the control qubit is in state $|1\rangle$. In the corresponding symbol (Figure 8.6) for the "wiring diagram" of a quantum computer performing the quantum Fourier transform, the upper wire denotes the target qubit, the lower wire the control qubit, and data are processed from left to right as usual.

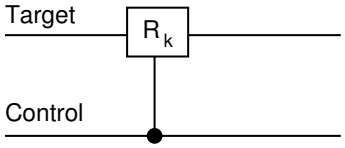

Figure 8.6: The controlled-$\mathbf{R}_k$ gate.

The controlled-$\mathbf{R}_k$ gate (for various k values) and the Hadamard gate are sufficient for the quantum Fourier transform circuit shown in Figure 8.7.

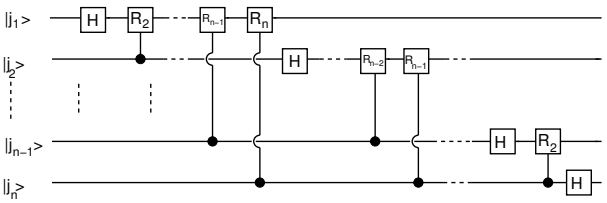

Figure 8.7: A circuit for the quantum Fourier transform. Not shown are the swap gates necessary to rearrange the output into the desired form.

To analyze how the circuit of Figure 8.7 performs the quantum Fourier transform, consider the input state $|j_1 j_2 \ldots j_n\rangle$. The Hadamard gate applied to the first qubit generates the state

$$2^{-1/2} \left(|0\rangle + e^{2\pi i 0.j_1} |1\rangle\right) |j_2 \ldots j_n\rangle,$$

since $e^{2\pi i 0.j_1} = (-1)^{j_1}$. The controlled-$\mathbf{R}_2$ gate produces

$$2^{-1/2} \left(|0\rangle + e^{2\pi i 0.j_1 j_2} |1\rangle\right) |j_2 \ldots j_n\rangle,$$

and the following controlled-**R** gates keep appending bits to the exponent of the phase factor of $|1\rangle_1$, leading finally to

$$2^{-1/2}\left(|0\rangle + e^{2\pi i 0.j_1 j_2 \ldots j_n}|1\rangle\right)|j_2 \ldots j_n\rangle.$$

The second qubit is treated in a similar way. The Hadamard gate generates

$$2^{-1/2}\left(|0\rangle + e^{2\pi i 0.j_1 j_2 \ldots j_n}|1\rangle\right) \cdot \left(|0\rangle + e^{2\pi i 0.j_2}|1\rangle\right)|j_3 \ldots j_n\rangle$$

and the controlled-$\mathbf{R}_2$ through $\mathbf{R}_{n-1}$ gates take care of the lower-order bits in the exponent of the phase factor of $|1\rangle_2$, leading to

$$2^{-1/2}\left(|0\rangle + e^{2\pi i 0.j_1 j_2 \ldots j_n}|1\rangle\right) \left(|0\rangle + e^{2\pi i 0.j_2 \ldots j_n}|1\rangle\right)|j_3 \ldots j_n\rangle.$$

Continuing this process we obtain the final state

$$2^{-\frac{n}{2}}\left(|0\rangle + e^{2\pi i 0.j_1 j_2 \ldots j_n}|1\rangle\right) \left(|0\rangle + e^{2\pi i 0.j_2 \ldots j_n}|1\rangle\right) \cdots \left(|0\rangle + e^{2\pi i 0.j_n}|1\rangle\right).$$

This is almost the desired result, except for the order of the qubits which can be rearranged by SWAP gates. This is similar to the classical FFT, which also requires some rearrangement of the numbers.

The total number of operations (gates) for the quantum Fourier transform is easily counted. The first qubit is acted on by a Hadamard gate and $n-1$ controlled-**R** gates, a total of n gates. The next qubit needs one controlled-**R** gate less, and so on. The total number of gates shown (implicitly) in Figure 8.7 thus is $n+(n-1)+\cdots+1 = n(n+1)/2$. In addition one needs about $n/2$ SWAP gates. The quantum Fourier transform thus needs of the order of n^2 gates (operations) to Fourier transform 2^n input data. This is much better than even the FFT which needs $\mathcal{O}(n2^n)$ steps, as discussed above. Note, however, that it is not possible to get out *all* of the amplitudes of the final state of the quantum Fourier transform, nor is it possible to efficiently prepare the input state for arbitrary amplitudes. This restricts application of the QFT to a special class of applications, such as the Shor algorithm.

8.4 The Grover algorithm: Looking for a needle in a haystack

8.4.1 (Un)structured databases

Grover's[3] algorithm [131, 132] is useful for a search in an unstructured database. This is a very important problem in data processing because every database is an unstructured one if the problem does not fit to the original design structure of the data base. Typical databases are organized in a way that makes it easy to find some entries (e.g. if they are listed alphabetically), but much harder for others.

SuperBioMarkt AG Harkort-18	7 77 09 95
SuperBioMarkt AG Westfalendamm 285	94 14 43 24
SuperVistaAG Kamp-30	18 99 99 73
Suplicki Brigitte Frankfurter-16	51 07 89
- Olaf Donner-24	2 17 68 30
Suppa Antonio	1 37 77 70
Supply Olaf Kiosk Echeloh 60	1 50 38 94
- Olaf Echeloh 60	7 26 56 00
Supren GmbH Joseph-von-Fraunhofer-20	9 70 03 90

Figure 8.8: Short section of a phone book (Dortmund).

As an example, Figure 8.8 shows a short section of a phone book from Dortmund. Entries follow alphabetic ordering of names, but the corresponding phone numbers appear completely random. If one searches for the phone number of a person, one therefore needs to check at most $\log N$ entries, where N is the total number of entries. However, if one needs the name of the person with a specific phone number, this will require checking $N/2$ entries on average. Grover's algorithm reduces the number of calls to $\mathcal{O}(\sqrt{N})$, which is a significant reduction for large N.

In this section we will not deal with the practical implementation of Grover's algorithm, that is, how to couple an existing classical database to this quantum algorithm, etc. We will only outline how this beautiful algorithm allows the

[3] Lov Kumar Grover (born 1961)

solution to "grow" out of the noise by iterating a simple procedure. As with all growing things, however, it is important to do the harvesting at the right time. It turns out that the same procedures can be used to grow the solution and to determine the time for the harvest.

For an implementation of Grover's algorithm employing NMR techniques, see [140]. An interesting implementation of Grover's algorithm based purely on the Fourier transforming capabilities of classical wave optics has also been demonstrated, see [141].

8.4.2 Oracle functions

Let the search space of our problem have N elements (entries in the phone directory, in the introductory example), indexed 0 to $N-1$, and for simplicity, $N = 2^n$. Let the search problem have M solutions (persons living at the given street address). The solutions can be characterized by some function f with the property

$$f(x) = \begin{cases} 1 & \text{if } x \text{ is a solution} \\ 0 & \text{if } x \text{ is not a solution.} \end{cases} \tag{8.17}$$

We are able, by some kind of "detector" to recognize a solution if we are confronted with the x^{th} element of the database. In our example this is simple: we just check the item "street address" in the telephone directory entry number x and output a 1 if it fits and a zero otherwise. In other examples this step may be much more complicated. Grover's algorithm minimizes the number of calls to this "detector" function, or *oracle* function as it is commonly called.

Like other functions, the oracle function corresponds in a quantum algorithm to a unitary operator $\mathbf{O}$. This operator acts on the tensor product of the quantum register holding the index x and a single oracle qubit $|q\rangle$ in the following way:

$$\mathbf{O}|x\rangle|q\rangle = |x\rangle|q \oplus f(x)\rangle,$$

that is, the oracle qubit is flipped when the database item with the number x is a solution of the search problem. If we initialize the oracle qubit in the state

$$|q_0\rangle := \frac{|0\rangle - |1\rangle}{\sqrt{2}},$$

application of the quantum oracle will lead to

$$\mathbf{O}|x\rangle|q_0\rangle = (-1)^{f(x)}|x\rangle|q_0\rangle.$$

Here, the oracle qubit is not changed, and in fact remains in its initial state during the whole calculation. We will henceforth omit it from our calculations (without forgetting that it is needed). So from now on we will abbreviate the above equation in the following way:

$$\mathbf{O}|x\rangle = (-1)^{f(x)}|x\rangle.$$

The oracle *marks* the solutions of the search problem by a minus sign. We will see that only $\mathcal{O}\left(\sqrt{N/M}\right)$ calls to the quantum oracle are necessary to solve the search problem. We wish to stress again that the oracle does not by some magic *know* the solution, it is only able to *recognize* if a candidate is a solution. Think of the prime factoring problem to note the difference: it is easy to *check* if a proposed candidate divides a number. An appropriate circuit performing test divisions would be used as an oracle in that case.

8.4.3 The search algorithm

The key point of the search algorithm will be to use the phase factors (minus signs) marking the solutions to let the amplitudes of the solution states grow out of the set of all possible states, and to "harvest" them at the right time, as noted above. We will now first list the steps of the search algorithm and then analyze what these steps do.

Step 1. Initialize the n-qubit index register

$$|\psi_1\rangle = |\vec{0}\rangle.$$

(All n qubits are set to their $|0\rangle$ states.)

Step 2. Apply the Hadamard transform

$$|\psi_2\rangle = \mathbf{H}^{\otimes n}|\vec{0}\rangle = N^{-1/2}\sum_{x=0}^{N-1}|x\rangle \quad (N = 2^n),$$

to generate an equal-weight, equal-phase superposition of all computational basis states.
Steps 3 and following. Iterate with the Grover operator **G**

$$|\psi_{k+1}\rangle = \mathbf{G}|\psi_k\rangle,$$

where the Grover operator consists of four substeps, which are shown in Figure 8.9.

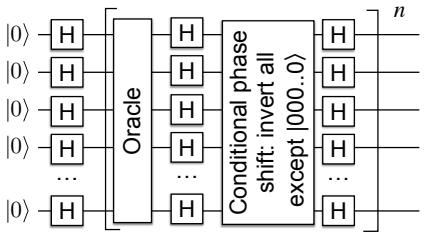

Figure 8.9: Network diagram of the 4 step Grover operator.

Sub-step 1. Apply the oracle

$$|\psi_{k+1/4}\rangle = \mathbf{O}|\psi_k\rangle$$

(we use fractional indices to symbolize that these are sub-steps of the Grover iteration step).
Sub-step 2. Apply the Hadamard transform

$$|\psi_{k+1/2}\rangle = \mathbf{H}^{\otimes n}|\psi_{k+1/4}\rangle.$$

Sub-step 3. Apply a conditional π phase shift, that is, reverse the signs of all computational basis states except $|\vec{0}\rangle$:

$$|\psi_{k+3/4}\rangle = \mathbf{C}_\pi|\psi_{k+1/2}\rangle,$$

where

$$\mathbf{C}_\pi|x\rangle = (-1)^{\delta_{x0}-1}|x\rangle.$$

Sub-step 4. Apply the Hadamard transform again

$$|\psi_{k+1}\rangle = \mathbf{H}^{\otimes n}|\psi_{k+3/4}\rangle.$$

Sub-steps 2,3, and 4 can be efficiently implemented on a quantum computer: remember that $\mathbf{H}^{\otimes n}$ creates 2^n states (in a superposition) with just n operations; conditional phase shifts are also easy to construct from a complete set of quantum gates. The oracle *may* be computationally expensive, but we use it only once per iteration step.

8.4.4 Geometrical analysis

Let us analyze what the Grover iteration step does, other than calling the oracle. The conditional phase shift may be written as

$$\mathbf{C}_\pi = -\mathbf{1} + 2|\vec{0}\rangle\langle\vec{0}|,$$

where $\mathbf{1}$ is the n-qubit unit operator and $|\vec{0}\rangle\langle\vec{0}|$ is the projection operator onto the basis state $|\vec{0}\rangle$. We know already that

$$\mathbf{H}^{\otimes n}|\vec{0}\rangle = |\psi_2\rangle(\text{ and } \langle\psi_2| = \langle\vec{0}|\mathbf{H}^{\otimes n}),$$

where $|\psi_2\rangle$ is the equal-weight (and equal-phase) superposition. Therefore

$$\begin{aligned}\mathbf{H}^{\otimes n}\mathbf{C}_\pi\mathbf{H}^{\otimes n} &= \mathbf{H}^{\otimes n}(2|\vec{0}\rangle\langle\vec{0}| - \mathbf{1})\mathbf{H}^{\otimes n} \\ &= (2|\psi_2\rangle\langle\psi_2| - \mathbf{1})\end{aligned}$$

and the Grover operator can be written as

$$\mathbf{G} = \mathbf{H}^{\otimes n}\mathbf{C}_\pi\mathbf{H}^{\otimes n}\mathbf{O} = (2|\psi_2\rangle\langle\psi_2| - \mathbf{1})\,\mathbf{O}.$$

This operation has a nice algebraic interpretation: the amplitudes of the computational basis states are "inverted about their average" (or mean) as is often said. However, we will not employ this algebraic interpretation (which is explained in Chapter 6 of [40]), because it turns out that there is an even nicer geometrical interpretation.

The Grover iteration corresponds to a rotation in the two-dimensional space spanned by the starting vector $|\psi_2\rangle$ (the uniform superposition of *all* basis states) and the uniform superposition of the states corresponding to the M solutions of the search problem, and we will see that the rotation moves the state into the right direction.

To see this we define two normalized states:

$$\begin{aligned}|\alpha\rangle &= \frac{1}{\sqrt{N-M}}\sum_x (1 - f(x))|x\rangle \\ |\beta\rangle &= \frac{1}{\sqrt{M}}\sum_x f(x)|x\rangle,\end{aligned}$$

with the function $f(x)$ defined by (8.17). Obviously $|\beta\rangle$ is the uniform superposition of the desired states and $|\alpha\rangle$ that of the remaining states.

We can then write the state $|\psi_2\rangle$ in the search algorithm as a superposition of $|\alpha\rangle$ and $|\beta\rangle$:

$$\begin{aligned} |\psi_2\rangle &= \sqrt{\frac{N-M}{N}}|\alpha\rangle + \sqrt{\frac{M}{N}}|\beta\rangle \\ &= \cos\frac{\theta}{2}|\alpha\rangle + \sin\frac{\theta}{2}|\beta\rangle, \end{aligned}$$

which defines the angle θ. Now recall that the oracle marks solutions of the search problem with a minus sign such that

$$\mathbf{O}|\psi_2\rangle = \cos\frac{\theta}{2}|\alpha\rangle - \sin\frac{\theta}{2}|\beta\rangle.$$

The $|\beta\rangle$ component of the initial state thus gets reversed, whereas the $|\alpha\rangle$ component remains the same. In the $|\alpha\rangle, |\beta\rangle$ plane this is a *reflection* about the $|\alpha\rangle$ axis. (See Figure 8.10.)

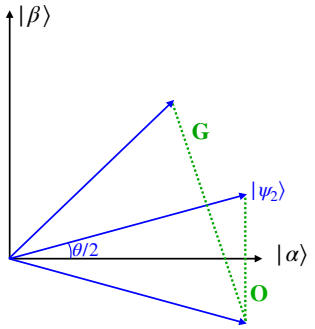

Figure 8.10: The Grover iteration as a twofold reflection, or a rotation (see text for details).

The remaining three sub-steps of $\mathbf{G}$ perform another reflection:

$$\begin{aligned} 2|\psi_2\rangle\langle\psi_2| - \mathbf{1} &= |\psi_2\rangle\langle\psi_2| - (\mathbf{1} - |\psi_2\rangle\langle\psi_2|) \\ &= \mathbf{P}_2 - \mathbf{P}_2^{\perp}, \end{aligned}$$

where $\mathbf{P}_2$ is the projector onto the initial state $|\psi_2\rangle$ and $\mathbf{P}_2^{\perp}$ is the projector onto the subspace perpendicular to $|\psi_2\rangle$. The component perpendicular to $|\psi_2\rangle$ thus gets reversed so that we have performed a *reflection about* $|\psi_2\rangle$. A look at Figure 8.10 tells us that we have reached the state

$$\mathbf{G}|\psi_2\rangle = \cos\frac{3\theta}{2}|\alpha\rangle + \sin\frac{3\theta}{2}|\beta\rangle,$$

that is, $\mathbf{G}$ has performed a θ rotation.

Iteration of the Grover operator yields

$$\mathbf{G}^k|\psi_2\rangle = \cos\frac{2k+1}{2}\theta|\alpha\rangle + \sin\frac{2k+1}{2}\theta|\beta\rangle,$$

and we only have to choose k such that the $|\beta\rangle$ component is as large as possible. Measurement in the computational basis will then, with high probability, produce one of the components of $|\beta\rangle$, the solutions of the search problem.

For a detailed description of the search algorithm in a space with four states (admittedly not too large), see [40] or the popular article [142] by Grover.

How often do we have to apply the Grover operator? From Figure 8.10 and the definition of the angle θ we see that the necessary number of iterations is the closest integer (abbreviated CI) to $\frac{\pi-\theta}{2\theta}$,

$$\begin{aligned} R: &= \mathrm{CI}\left(\frac{\pi}{2\theta} - \frac{1}{2}\right) = \mathrm{CI}\left(\frac{\pi}{4\arcsin\sqrt{\frac{M}{N}}} - \frac{1}{2}\right) \\ &\leq \frac{\pi}{4}\sqrt{\frac{N}{M}} \end{aligned}$$

since $\arcsin x > x$. This moves the state quite close to the desired one: as each Grover iteration rotates the state by θ, we end up at most $\theta/2$ away from $|\beta\rangle$. For the interesting case $\frac{M}{N} \ll 1$ the error probability (given by the square of the $|\alpha\rangle$ component in the final state) is

$$p \leq \sin^2\frac{\theta}{2} = \frac{M}{N}.$$

Important remarks:

- Iterating more than R times worsens the result.
- In this version of the algorithm, it is necessary to know M, the number of solutions. If this information is not known a priory, the Grover operator can provide it.

8.4.5 Quantum counting

Here we discuss how the number M of solutions to the search problem can be counted by a quantum algorithm involving the Grover operator $\mathbf{G}$ again. The idea is simple: recall that in a suitable two-dimensional subspace, $\mathbf{G}$ is just a rotation and the rotation angle is related to M. This rotation angle can be determined by quantum Fourier transform techniques.

The rotation matrix for $\mathbf{G}$ in the basis $(|\alpha\rangle, |\beta\rangle)$ is

$$\mathbf{G} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}.$$

The eigenvectors of this matrix are $\frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ \pm i \end{pmatrix}$ with eigenvalues $e^{\pm i\theta}$. Recall that $\sin\frac{\theta}{2} = \sqrt{\frac{M}{N}}$. (Some problems may arise if $M > N/2$, because then $\theta > \pi/2$; however, these problems may always be circumvented by enlarging the search space from N to $2N$ by adding some fictitious directions to the Hilbert space, as discussed in [40]. We will ignore these problems altogether for simplicity.)

The problem of (approximately) counting the number M of solutions is thus reduced to estimating the phase θ of the unitary operator $\mathbf{G}$, the Grover gate. This task of *phase estimation* is very similar to the task of *period-finding* involved in Shor's algorithm as discussed in Section 8.3.4.

We can reformulate the problem: We consider a unitary operator $\mathbf{U}$ acting on one of its eigenvectors $|u\rangle$:

$$\mathbf{U}|u\rangle = e^{2\pi i\phi}|u\rangle,$$

where ϕ (between 0 and 1) is to be estimated. We further assume that we have access to "black boxes" for

- preparing $|u\rangle$,
- performing controlled-$\mathbf{U}^{(2^j)}$ operations $(j = 0, 1, ..)$.

We then need an algorithm for measuring ϕ.

8.4.6 Phase estimation

The phase estimation algorithm needs two registers. The first register contains t qubits, initially all in the state $|0\rangle$ (t depending on the demanded *accuracy* and *success probability* of the algorithm). The second register holds the state $|u\rangle$ initially.

The algorithm works as follows.
Step 1. Apply the Hadamard transform $\mathbf{H}^{\otimes t}$ to the first register, to generate the state

$$\mathbf{H}^{\otimes t}|\vec{0}\rangle = \frac{1}{\sqrt{2^t}}\sum_{x=1}^{2^t}|x\rangle,$$

which is the by now well-known equal-weight, equal-phase superposition.
Step 2.k ($k = 0, ..., t-1$). Apply the controlled-$\mathbf{U}^{(2^k)}$ operation to register 2, using qubit k of the first register as control qubit. This puts register 2 in state

$|u\rangle$ if qubit k is $|0\rangle$

and in state

$e^{2\pi i 2^k\phi}|u\rangle$ if qubit k is $|1\rangle$.

Register 2 therefore stays in the state $|u\rangle$ all the time, up to phase factors which we can collect next to the qubits of register 1 which control them. The state of the first register thus can be written

$$\begin{aligned} &\frac{1}{2^{t/2}}\left(|0\rangle + e^{2\pi i 2^{t-1}\phi}|1\rangle\right)\left(|0\rangle + e^{2\pi i 2^{t-2}\phi}|1\rangle\right) \\ &\cdots\left(|0\rangle + e^{2\pi i 2^0\phi}|1\rangle\right) \\ = \;& \frac{1}{2^{t/2}}\sum_{k=0}^{2^{t-1}} e^{2\pi i\phi k}|k\rangle. \end{aligned}$$

The second register remains in state $|u\rangle$ and is not written here. For ease of discussion, assume that ϕ is a t-bit binary fraction, $\phi = 0\,.\,\phi_1\phi_2\ldots\phi_t$ (remember $\phi \leq 1$). The state of register 1 is just

$$\begin{aligned} &\frac{1}{2^{t/2}}\left(|0\rangle + e^{2\pi i\,0\,.\,\phi_t}|1\rangle\right)\left(|0\rangle + e^{2\pi i\,0\,.\,\phi_{t-1}\phi_t}|1\rangle\right) \\ &\cdots\left(|0\rangle + e^{2\pi i\,0\,.\,\phi_1\phi_2\ldots\phi_t}|1\rangle\right) \end{aligned}$$

since $e^{2\pi i m} = 1$ for integer m.

We now recall the discussion of the quantum Fourier transform from Shor's algorithm in Section (8.3.6). There we constructed a quantum circuit performing the quantum Fourier transformation

$$|j_1 \dots j_n\rangle \mapsto \frac{1}{2^{n/2}} \left(|0\rangle + e^{2\pi i\, 0\,.\, j_n}|1\rangle\right) \left(|0\rangle + e^{2\pi i\, 0\,.\, j_{n-1} j_n}|1\rangle\right) \cdots \left(|0\rangle + e^{2\pi i\, 0\,.\, j_1 j_2 \dots j_n}|1\rangle\right).$$

8.4.7 Reading the Phase

The *inverse* quantum Fourier transform can be performed by simply reversing the QFT circuit. Applying the inverse QFT to the state of register 1 leads to the state

$$|\phi_1 \dots \phi_t\rangle \tag{8.18}$$

and therefore we can measure ϕ exactly in this example, where ϕ has exactly t bits. If the binary expansion of ϕ is longer than t bits, for example, if ϕ is irrational so that its binary expansion does not terminate at all, only an estimate is possible. In that case the algorithm does not uniquely lead to the single basis state (8.18) but to a superposition of basis states with probabilities strongly concentrated on t-bit binary fractions ϕ' approximating ϕ. The probabilities discussed in Section 8.3.4 for the period estimate are very similar to this clustering: determining the period is essentially equivalent to the phase estimate which we are presently discussing.

To determine the reliability of this algorithm, we assume that we want to achieve a certain precision δ in estimating ϕ. The probability of failure of the algorithm is then the cumulative probability of all states with $|\phi' - \phi| > \delta$. That probability can be estimated, see Section 5.2.1 of [40]. It turns out that if t qubits are available, an n-bit approximation to ϕ may be found with probability of success at least $1 - \varepsilon$, if

$$t = n + \operatorname{int} \log_2 \left(1 + \frac{1}{4\varepsilon}\right). \tag{8.19}$$

Here, int denotes the integer part of a real number, as usual.

An important point that remains is the preparation of the eigenstate $|u\rangle$. In the worst case we are not able to prepare a specific eigenstate, but only some state $|\psi\rangle$ which can then be expanded in $\mathbf{U}$-eigenstates,

$$|\psi\rangle = \sum_u c_u |u\rangle, \text{ where } \mathbf{U}|u\rangle = e^{2\pi i \phi_u}|u\rangle.$$

Running the phase estimation algorithm with input $|\psi\rangle$ in the second register leads (due to linearity) to the output

$$\sum_u c_u |\phi'_u\rangle |u\rangle,$$

where ϕ'_u is an approximation to the phase ϕ_u. We thus obtain the possible phase values of $\mathbf{U}$ with their respective probabilities $|c_u|^2$ as given by the initial state.

In the special case of the Grover algorithm it turns out that we are lucky. Recall that the starting vector of the Grover algorithm was a combination of $|\alpha\rangle$ and $|\beta\rangle$, or equivalently, of the two eigenstates of the unitary operator $\mathbf{G}$ (the Grover operator) so that the phase estimation algorithm will give us approximations to either θ or $(2\pi) - \theta$ with both of which we will be content, because knowing θ will enable us to optimize the number of iterations of $\mathbf{G}$ and therefore find a solution of the search problem with high probability.

We will not discuss how to *really* search an unstructured data base etc., and we will also not go into the detailed performance and probability estimates. Some remarks on these topics may be found in Chapter 6 of [40], and some generalizations and references to interesting applications are in [130].

8.5 Other algorithms

8.5.1 Gradient estimation

Finding the gradient $\vec{\nabla} f$ of a function is important for many applications involving optimiza-

tion tasks, such as quantum mechanical calculations, machine learning, or artificial intelligence. Numerically finding the gradient of a function $f(\vec{x})$ of d variables x_i requires on a classical computer the evaluation of f for $d+1$ arguments.

In 2005, Jordan proposed a quantum algorithm [143] that achieves this task with a single function evaluation, called the Quantum algorithm for Numerical Gradient Estimation (QNGE). It was demonstrated experimentally the following year [107]. Similar schemes can be used to solve systems of equations [144], including nonlinear differential equations [145] or the solution of quantum chemical problems [146, 147]. This procedure is also an interesting candidate for many optimization tasks [148, 149], such as quantum machine learning, where quantum computers are used to accelerate machine learning schemes [150].

We consider a real scalar function f of d real variables, $f : R^d \to R$. The goal is to compute the gradient $\vec{\nabla} f$ numerically, with the smallest possible number of calls of the function f. For simplicity, we write it for the 1-dimensional case, where

$$\nabla f = \frac{f(\ell/2) - f(-\ell/2)}{\ell}$$

and ℓ represents the (small) interval between the points where the function is evaluated. On a classical computer this requires a minimum of $d+1$ function calls.

For the quantum algorithm, we need d binary strings, each of length n, where n depends on the desired precision. In addition, an ancilla register is required for the output, which is first initialized to 1. As usual in quantum function evaluation, the black box implements the function call by adding (modulo 2) the result to the output register. Real numbers are represented as

$$x = \frac{\ell}{N-1}\left(\delta - \frac{N-1}{2}\right).$$

Figure 8.11 shows a block diagram of the algorithm. To implement the algorithm, one first

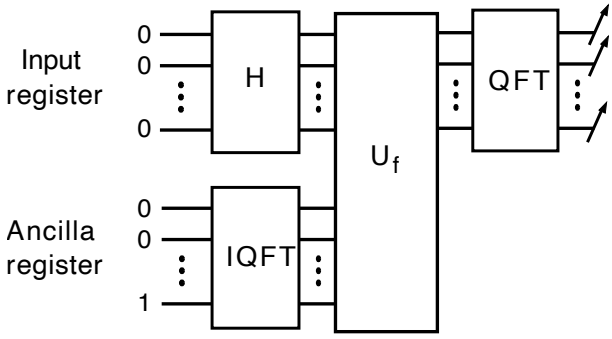

Figure 8.11: Network representation of the quantum numerical gradient estimation algorithm.

performs a Hadamard transform to the input register to create a uniform superposition of the states, and an inverse Fourier transform on the ancilla register to generate a plane wave state

$$\frac{1}{\sqrt{N_0}} \sum_{k=0}^{N_0-1} e^{-i(2\pi k/N_0}|k\rangle.$$

When the function evaluation is performed, the result is added to the plane wave state, interfering with it. This directly generates a binary representation of the gradient in the output state. The input state is unchanged.

For an actual implementation, some additional points must be considered. As an example, the expected range of function values should be known for properly choosing the number of required qubits and some factors for the encoding / decoding.

8.5.2 Image processing

Image processing is a major task in many contexts. Accordingly, processing images efficiently has an enormous potential. Figure 8.12 compares the classical with the quantum image processing scheme. In the latter, the pixel data are encoded in the amplitudes of the quantum state: if we write the vector $\vec{f} = [F_{11}, F_{21}, \dot{F_{ML}}]$, we can encode the pixel values (suitably scaled) in the coefficients of the computational basis states,

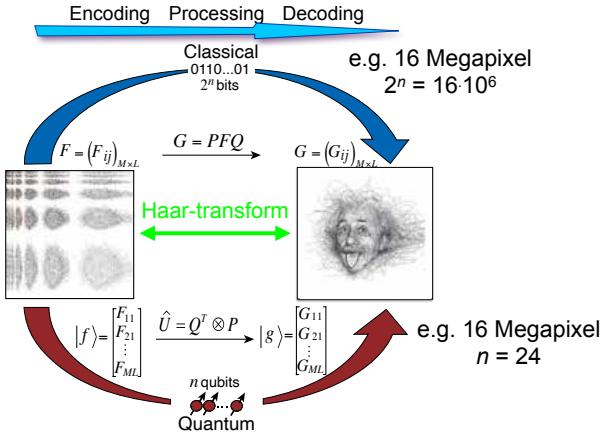

Figure 8.12: Comparison of classical and quantum image processing.

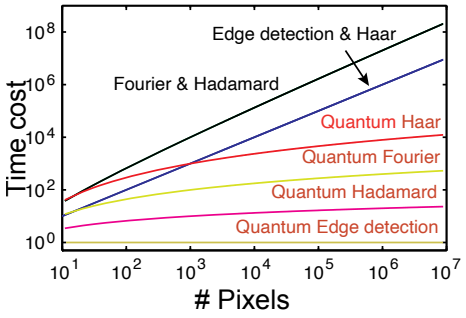

Figure 8.14: Number of computational steps required for different classical and quantum algorithms as a function of image size.

$|\vec{f}\rangle = \sum_k c_k |k\rangle$ as

$$c_k = F_{i,j} / \sqrt{\sum F_{i,j}^2}. \tag{8.20}$$

For certain tasks, very large gains are possible, both in terms of storage space, as well as in terms of time cost.

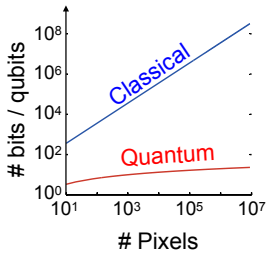

Figure 8.13: Storage space required for a b/w picture in a classical computer vs. a quantum computer using the encoding scheme of eq. (8.20).

Figure 8.13 compares the requirements for the storage space required for a classical computer vs. a quantum computer. A classical computer requires Nd bits of storage for the representation of an N pixel image if each pixel has a depth of d bits. Encoding the same image into a quantum state reduces the requirement to $\log N$ qubits.

Similarly, the time cost for various processing tasks can be reduced qualitatively. Figure 8.14 compares the number of computational steps required for the execution of some typical computational procedures used in image processing. While the classical algorithms scale at least linearly with the number of pixels, the quantum algorithms scale logarithmically. The quantum edge detection algorithm requires only a single step, independent of the picture size. This algorithm consists in finding the the boundaries between two areas of different colors. This is a very important task, for which our brain (and other biological computers) have developed specialized hardwired circuits. Classical algorithms for edge detection in an $M \times L$ image require of the order $M \times L$ operations involving the comparison of neighboring pixel values. Using the encoding (8.20) and applying a Hadamard gate to the lsb, the output state contains sums and differences of neighboring pixel values c_i:

$$|g\rangle = |c_0{+}c_1, c_0{-}c_1, \ldots, c_{N-2}{+}c_{N-1}, c_{N-2}{-}c_{N-1}\rangle.$$

i.e. it now contains sums and differences between neighboring pixel values. The non-zero differences therefore encode the boundary. Since only a single Hadamard gate is needed, independent of the image size, this is a highly efficient algorithm.[151]

8.5.3 Machine learning and reservoir computing

Machine learning (ML) uses statistical analysis of large sets of data for making predictions about similar data sets. Typical applications are pattern recognition and the dynamics of complex systems, such as weather forecast. In the training phase, it adjusts the values of a large set of parameters to minimize the difference between its predictions and the known solutions to the training data.[4] As shown in Figure 8.15, this is an important part of the wider field of artificial intelligence (AI). The optimization of the parameter sets relies on the evaluation of gradients to minimize the deviations.

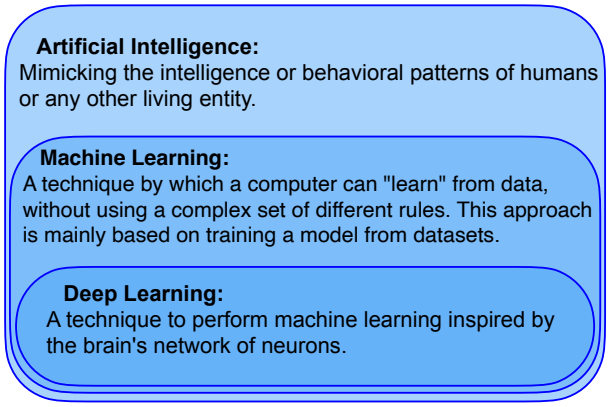

Figure 8.15: Machine Learning as a tool for artificial intelligence.

Since this training involves large computational efforts, it relies on specialized hardware - either graphical processing units (GPUs) or artificial neural networks: these are interconnected groups of nodes inspired by the architecture of the human brain. In these networks, the signals are typically sent through multiple layers of neurons from the input to the output layer. An extension of this is reservoir computing; it maps input signals into higher dimensional computational spaces through the dynamics of a fixed, non-linear system called a reservoir [152]. If a quantum mechanical system is used for this, we arrive at the field of quantum reservoir computing.

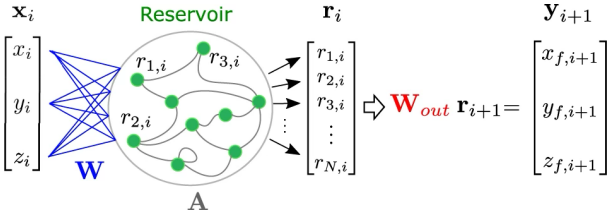

Figure 8.16: Principle of reservoir computing.

Figure 8.16 shows the principle of reservoir computing. The reservoir represents a fixed subsystem that converts input into output in a predefine way, which is in general nonlinear. It can therefore effectively simulate a nonlinear dynamics or a higher-dimensional system. It has been tested for a number of applications like pattern recognition or forecasting of time series.

8.6 Quantum simulations

Most of the current work on implementations of quantum algorithms concentrates on the algorithms for factoring (Shor) and database searching (Grover). An important reason may be that these algorithms are related to important tasks with which many potential users are familiar. From a physics perspective, however, the application of quantum processors for simulating quantum mechanical systems offers a more exciting potential of opening completely unexplored new areas.

8.6.1 Potential and limitations

One of the earliest motivations for the development of quantum computers was the suggestion by Feynman [14] (see Section 1.3.1 and Chapter 6) that quantum processors may be the only possibility for efficiently simulating quantum mechanical systems. While general purpose quantum computers that are sufficiently powerful for exactly solving the Schrödinger equation of realistic quantum systems like molecules or crystals

[4] In unsupervised learning, the system tries to find pattern without prior knoledge.