Quantum information theory (20110401)

Lecturer: Jens Eisert

Chapter 9: Quantum algorithms

Contents

Chapter 9

Quantum algorithms

9.1 Deutsch and Deutsch-Jozsa algorithms

In the same scientific paper in which David Deutsch introduced the notion of the universal quantum computer, he also presented the first quantum algorithm [1]. The problem that this algorithm addresses, later referred to as Deutsch's problem, is a very simple one. Yet the *Deutsch algorithm* already exemplifies the advantages of a quantum computer through skillfully exploiting quantum parallelism. Like the Deutsch algorithm, all other elementary quantum algorithms in this section amount to deciding which *black box* out of finitely many alternatives one has at hand. Such a black box is often also referred to as *oracle*. An input may be given to the oracle, one may read out or use the outcome in later steps of the quantum algorithm, and the objective is to find out the functioning of the black box. It is assumed that this oracle operation can be implemented with some sequence of quantum logic gates. The complexity of the quantum algorithm is then quantified in terms of the number of queries to the oracle.

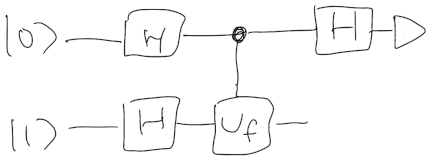
9.1.1 Deutsch algorithm

With the help of this algorithm, it is possible to decide whether a function has a certain property with a single call of the function, instead of two calls that are necessary classically. Let

$$f : \{0, 1\} \longrightarrow \{0, 1\} \tag{9.1}$$

be a function that has both a one-bit domain and range. This function can be either *constant* or *balanced*, which means that either $f(0) \oplus f(1) = 0$ or $f(0) \oplus f(1) = 1$ holds. The problem is to find out with the minimal number of function calls whether this function f is constant or balanced. In colloquial terms, the problem under consideration may be described as a procedure to test whether a coin is fake (has two heads or two tails) or a genuine coin.

[1] Quantum Turing machines were first considered by Benioff and developed by Deutsch.



Classically, it is obvious that two function calls are required to decide which of the two allowed cases is realised, or, equivalently, what the value of $f(0) \oplus f(1)$ is. A way to compute the function f on a quantum computer is to transform the state vector of two qubits according to

$$|x, y\rangle \mapsto U_f|x, y\rangle = |x, f(x) \oplus y\rangle. \tag{9.2}$$

In this manner, the evaluation can be realized unitarily. The above map is what is called a standard quantum oracle (as opposed to a minimal quantum oracle, which would be of the form $|x\rangle \mapsto |f(x)\rangle$). The claim now is that using such an oracle, a single function call is sufficient for the evaluation of $f(0) \oplus f(1)$. In order to show this, let us assume that we have prepared two qubits in the state with state vector

$$|\Psi\rangle = (H \otimes H)|0, 1\rangle, \tag{9.3}$$

where H denotes the Hadamard gate. We now apply the unitary U_f once to this state, and finally apply another Hadamard gate to the first qubit. The resulting state vector hence reads as

$$|\Psi'\rangle = (H \otimes \mathrm{id})U_f(H \otimes H)|0, 1\rangle. \tag{9.4}$$

A short calculation shows that $|\Psi'\rangle$ can be evaluated to

$$|\Psi'\rangle = \pm|f(0) \oplus f(1)\rangle H|1\rangle. \tag{9.5}$$

The second qubit is in the state corresponding to the vector $H|1\rangle$, which is of no relevance to our problem. The state of the first qubit, however, is quite remarkable: encoded is $|f(0) \oplus f(1)\rangle$, and both alternatives are decidable with unit probability in a measurement in the computational basis, as the two state vectors are orthogonal[2]. That is, with a single measurement of the state, and notably, with a single call of the function f, of the first qubit we can decide whether f was constant or balanced.

9.1.2 Deutsch-Jozsa algorithm

The Deutsch algorithm has not yet any implication on superiority of a quantum computer as compared to a classical as far as the query complexity is concerned. After all, it is merely one function call instead of two. The situation is different in case of the extension of the Deutsch algorithm known as *Deutsch-Jozsa algorithm*. Here, the task is again to find out whether a function is constant or balanced, but f is now a function

$$f : \{0, 1\}^N \longrightarrow \{0, 1\}, \tag{9.6}$$

[2]Note that the presented algorithm is not quite the one in the original paper by Deutsch, which allowed for an inconclusive outcome in the measurement. This deterministic version of the Deutsch algorithm is due to Cleve, Ekert, Macchiavello, and Mosca.

where N is some natural number. It is promised that the function is either constant, which now means that either $f(i) = 0$ for all $i = 0, \ldots, 2^N - 1$ or $f(i) = 1$ for all i, or balanced. It is said to be balanced if the image under f takes as many times the value 1 as the value 0. The property to be balanced or constant can be said to be a global property of several function values. It is a promised property of the function, which is why the Deutsch-Jozsa algorithm is being classified as a *promise algorithm*. There are only two possible black boxes to the disposal, and the tasks is to find out which one is realised.

It is clear how many times one needs to call the function on a classical computer: the worst case scenario is that after $2^N/2$ function calls, the answer has been always 0 or always 1. Hence, $2^N/2 + 1$ function calls are required to know with certainty whether the function is balanced or constant (a result that can be significantly improved if probabilistic algorithms are allowed for). Quantum mechanically, again a single function call is sufficient. Similarly to the above situation, one may prepare $N + 1$ qubits in the state with state vector

$$|\Psi\rangle = H^{\otimes(N+1)}|0, \ldots, 0, 1\rangle, \tag{9.7}$$

and apply the unitary U_f as in Eq. (9.2) to it, acting as an oracle, and apply $H^{\otimes N} \otimes \mathrm{id}$ to the resulting state, to obtain

$$|\Psi'\rangle = (H^{\otimes N} \otimes \mathrm{id})U_f H^{\otimes(N+1)}|0, \ldots, 0, 1\rangle. \tag{9.8}$$

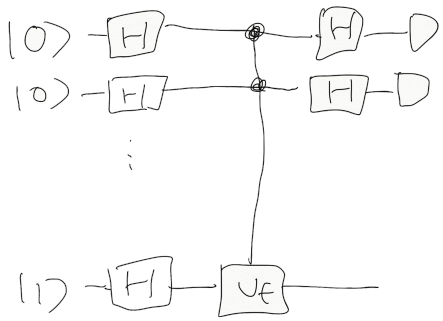
In the last step, one performs a measurement on the first N qubits in the computational basis. In effect, one observes that exactly if the function f is constant, one obtains the measurement outcome corresponding to $|0, 0, \ldots, 0\rangle$ with certainty. For any other output the function was balanced. So again, the test for the promised property can be performed with a single query, instead of $2^N/2 + 1$ classically [3].

After all, the performance of the Deutsch-Jozsa algorithm is quite impressive. If there is a drawback to this, yet, it is that unfortunately, the Deutsch-Jozsa algorithm is to some extent artificial in nature, and it lacks an actual practical application emerging in a natural context. The astonishing difference in the number of queries in the quantum and classical case also disappears if classically probabilistic algorithms are allowed for: in fact, using a probabilistic algorithm, a polynomial number of queries achieves an exponentially good success probability.

[3] There are a number of related problems that show very similar features. In the *Bernstein-Vazirani algorithm* again a function $f : \{0,1\}^N \longrightarrow \{0,1\}$ is given, promised to be of the form

$$f(x) = ax. \tag{9.9}$$

for $a, x \in \{0,1\}^N$ for some natural number N. ax denotes the standard scalar product $ax = a_0x_0 + \cdots + a_{2^N-1}x_{2^N-1}$. How many measurements are required to find the vector a of zeros and ones? Classically, one has to perform measurements for all possible arguments, and in the end solve a system of linear equations. With the standard oracle $|x, y\rangle \mapsto |x, f(x) \oplus y\rangle$ at hand, in its quantum version in the Bernstein-Vazirani algorithm only a single call of the oracle is required. Although it has been convincingly argued that one does not have to evoke the metaphor of quantum parallelism to interpret the functioning of the quantum computer in the Bernstein-Vazirani problem – the difference from quantum to classical lies rather in the ability to reverse the action of a CNOT gate by means of local operations on the control and target qubits – the surprisingly superior performance of the quantum algorithm to its classical counterpart is manifest.



9.1.3 Simon's algorithm

Simon's problem is an instance of an oracle problem which is classically hard, even for probabilistic algorithms, but tractable for quantum computers. The task is to find the period p of a certain function $f : \{0,1\}^N \longrightarrow \{0,1\}^N$, which is promised to be 2-to-1 with $f(x) = f(y)$ if and only if $y = x \oplus p$. Here, x and y denote binary words of length N, where $\oplus$ now means bitwise addition modulo 2. The problem can be stated as a decision problem as well and the goal would then be to decide whether or not there is a period, i.e., whether f is 2-to-1 or 1-to-1.

Classically the problem is hard, since the probability of having found two identical elements x and y after $2^{N/4}$ queries is still less than $2^{-N/2}$. Simon's quantum solution is now the following: start with a state vector $(H|0\rangle)^{\otimes N}|0\rangle^{\otimes N}$ and run the oracle once yielding the state vector $2^{-N/2}\sum_x |x\rangle|f(x)\rangle$. Then measure the second register[4]. If the measurement outcome is $f(x_0)$, then the state vector of the first register will be

$$\frac{1}{\sqrt{2}}\big(|x_0\rangle + |x_0 \oplus p\rangle\big). \tag{9.10}$$

Applying a Hadamard gate to each of the N remaining qubits leads to

$$\frac{1}{2^{(N+1)/2}} \sum_y \Big((-1)^{x_0 \cdot y} + (-1)^{(x_0 \oplus p)\cdot y}\Big)\,|y\rangle \tag{9.11}$$

$$= \frac{1}{2^{(N-1)/2}} \sum_{p\cdot y = 0} (-1)^{x_0 \cdot y}|y\rangle. \tag{9.12}$$

If we finally measure the first register in computational basis, we obtain a value y which is such that $y \cdot p = 0$ modulo 2. Repeating this procedure in order to get $N-1$ linearly independent vectors $y_1, \ldots, y_{N-1}$ we can determine p from the set of equations $\{y_i \cdot p = 0\}$. To this end we have to query the oracle $\mathcal{O}(N)$ times[5]. Hence, we get an exponential speed up compared to any classical algorithm. And in contrast to the Deutsch-Jozsa algorithm this exponential gap remains if we allow for probabilistic

[4]Note that this step is not even necessary – it is merely a pedagogical one.

[5]This symbol is the "big-O" Landau symbol for the asymptotic upper bound. In the rest of the chapter, this notation will be used even if the asymptotic behavior could be specified more precisely.

classical algorithms[6]. Simon's algorithm has much in common with Shor's algorithm: they both try to find the period of a function[7], both yield an exponential speed-up, and both make use of classical algorithms in a post processing step. Actually, Shor's work was inspired by Simon's result.

9.2 Grover's database search algorithm

The speed-up of the presented quantum algorithms for the Deutsch-Jozsa and Simon's problem is enormous. However, the oracle functions are constrained to comply with certain promises and the considered tasks hardly appear in practical applications. In contrast to that, Grover's algorithm deals with a frequently appearing problem: *database search.*

Assume we have an unsorted list and want to know the largest element, the mean, whether there is an element with certain properties or the number of such elements – all these are common problems or necessary subroutines for more complex programs and due to Grover's algorithm all these problems in principle admit a typically quadratic speed-up compared to classical solutions. Such an improvement of the performance might not appear very spectacular, however, the problems to which it is applicable are quite numerous[8] and the progress from the ordinary Fourier transform to the FFT already demonstrated how a quadratic speed-up in an elementary routine can boost many applications.

Consider the problem of searching a marked element $x_0 \in \{1, \ldots, N\}$ within an unsorted database of length $N = 2^n$. Whereas classically we have to query our database $\mathcal{O}(N)$ times in order to identify the sought element, Grover's algorithm will require only $\mathcal{O}(\sqrt{N})$ trials. Let the database be represented by a unitary[9]

$$U_{x_0} = \mathrm{id} - 2|x_0\rangle\langle x_0|, \tag{9.13}$$

which flips the sign of $|x_0\rangle$ but preserves all vectors orthogonal to $|x_0\rangle$. The first step of the algorithm is to prepare an equally weighted superposition of all basis states $|\Psi\rangle = \frac{1}{\sqrt{N}} \sum_x |x\rangle$. As we have already seen previously this can be achieved by applying N Hadamard gates to the state vector $|0\rangle$. Next, we apply the *Grover operator*

$$G = U_\Psi U_{x_0}\ , \quad U_\Psi = 2|\Psi\rangle\langle\Psi| - \mathrm{id} \tag{9.14}$$

to the state vector $|\Psi\rangle$. Geometrically, the action of G is to rotate $|\Psi\rangle$ towards $|x_0\rangle$ by an angle 2φ where $\sin\varphi = |\langle\Psi|x_0\rangle| = 1/\sqrt{N}$. The idea is now to iterate this rotation k-times until the initial state is close to $|x_0\rangle$, i.e.,

$$G^k|\Psi\rangle \approx |x_0\rangle. \tag{9.15}$$

[6] Simon's problem is an instance of an oracle problem relative to which BPP$\neq$BQP. That is, classical and quantum polynomial-time complexity classes for bounded error probabilistic algorithms differ relative to Simon's problem.

[7] Whereas Simon's problem is to find a period in $(\mathbb{Z}_2)^N$, Shor's algorithm searches for one in $\mathbb{Z}_{2^N}$.

[8] For instance, the standard solution to all NP-complete problems is doing an exhaustive search. Hence, Grover's algorithm would speed-up finding a solution to the travelling salesman, the Hamiltonian cycle and certain coloring problems.

[9] $|x_0\rangle\langle x_0|$ means the projector onto the vector $|x_0\rangle$. That is $U_{x_0}|x\rangle = (-1)^{\delta_{x,x_0}}|x\rangle$.

Measuring the system (in computational basis) will then reveal the value of x_0 with high probability.

So, how many iterations do we need? Each step is a 2φ-rotation and the initial angle between $|\Psi\rangle$ and $|x_0\rangle$ is $\pi/2 - \varphi$.[10] Using that for large N $\sin\varphi \approx \varphi$ we see that $k \approx \pi\sqrt{N}/4$ rotations will do the job and the probability of obtaining a measurement outcome different from x_0 will decrease as $\mathcal{O}(1/N)$. Since every step in the Grover iteration queries the database once, we need indeed only $\mathcal{O}(\sqrt{N})$ trials compared to $\mathcal{O}(N)$ in classical algorithms. To exploit this speed-up we need of course an efficient implementation not only of the database-oracle U_{x_0}, but also of the unitary U_Ψ. Fortunately, the latter can be constructed out of $\mathcal{O}(\log N)$ elementary gates.

What if there are more than one, say M, marked elements? Using the equally weighted superposition of all the respective states instead of $|x_0\rangle$ we can essentially repeat the above argumentation and obtain that $\mathcal{O}(\sqrt{N/M})$ queries are required in order to find one out of the M elements with high probability. However, performing further Grover iterations would be overshooting the mark: we would rotate the initial state beyond the sought target and the probability for finding a marked element would rapidly decrease again. If we initially do not know the number M of marked elements this is, however, not a serious problem. As long as $M \ll N$ we can still gain a quadratic speed-up by simply choosing the number of iterations randomly between 0 and $\pi\sqrt{N}/4$. The probability of finding a marked element will then be close to $1/2$ for every M. Notably, Grover's algorithm is optimal in the sense that any quantum algorithm for this problem will necessarily require $\mathcal{O}(\sqrt{N/M})$ queries.

9.3 Exponential speed-up in Shor's factoring algorithm

Shor's algorithm is without doubt not only one of the cornerstones of quantum information theory but also one of the most surprising advances in the theory of computation itself: a problem, which is widely believed to be *hard* becomes *tractable* by refereing to (quantum) physics – an approach completely atypical for the theory of computation, which usually abstracts away from any physical realization.

The problem Shor's algorithm deals with is *factorization*, a typical NP problem. Consider for instance the task of finding the prime factors of 421301. With pencil and paper it might probably take more than an hour to find them. The inverse problem, the multiplication 601×701, can, however, be solved in a few seconds even without having pencil and paper at hand[11]. The crucial difference between the two tasks multiplication and factoring is, however, how the degree of difficulty increases with the length of the numbers. Whereas multiplication belongs to the class of "tractable" problems for which the required number of elementary computing steps increases polynomially with the size of the input, every known classical factoring algorithm requires an exponentially increasing number of steps. This is what is meant by saying that factoring is an "intractable" or "hard" problem. In fact, it is this discrepancy between the complexity of the factoring problem and its inverse which is exploited in the most popular

[10] This clarifies why we start with the state vector $|\Psi\rangle$: the overlap $|\langle\Psi|x_0\rangle|$ does not depend on x_0.

[11] Actually, it takes eleven seconds for a randomly chosen Munich schoolboy at the age of twelve (the sample size was one).

public key encryption scheme based on RSA – its security heavily relies on the assumed difficulty of factoring. In a nutshell the idea of Shor's factoring algorithm is the following:

(i) *Classical part:* Using some elementary number theory one can show that the problem of finding a factor of a given integer is essentially equivalent to determining the period of a certain function.

(ii) *QFT for period-finding*: Implement the function from step (i) in a quantum circuit and apply it to a superposition of all classical input states. Then perform a discrete quantum Fourier transform (QFT) and measure the output. The measurement outcomes will be probabilistically distributed according to the inverse of the sought period. The latter can thus be determined (with certain probability) by repeating the procedure.

(iii) *Efficient implementation:* The crucial point of the algorithm is that the QFT as well as the function from step (i) can be efficiently implemented, i.e., the number of required elementary operations grows only polynomially with the size of the input. Moreover, the probability of success of the algorithm can be made arbitrary close to one without exponentially increasing effort.

Clearly, the heart of the algorithm is an efficient implementation of the QFT. Since Fourier transforms enter in many mathematical and physical problems one might naively expect an exponential speedup for all these problems as well. However, the outcome of the QFT is not explicitly available but "hidden" in the amplitudes of the output state, which can not be measured efficiently. Only global properties of the function, like its period, can in some cases be determined efficiently.

Nevertheless, a couple of other applications are known for which the QFT leads again to an exponential speed up compared to the known classical algorithms. The abstract problem, which encompasses all these applications is known as the "hidden subgroup problem" and another rather prominent representative of this type is the discrete logarithm problem. Let us now have a more detailed look at the ingredients for Shor's algorithm.

9.3.1 Classical part

Let N be an odd number we would like to factor and $a < N$ an integer which has no non-trivial factor in common with N, i.e., $gcd(N, a) = 1$. The latter can efficiently be checked by Euclid's algorithm[12]. A factor of N can then be found indirectly by determining the period p of the function $f : \mathbb{Z} \longrightarrow \mathbb{Z}_N$ defined as

$$f(x) = a^x \bmod N. \tag{9.16}$$

Hence, we are looking for a solution of the equation $a^p - 1 = 0 \bmod N$. Assuming p to be even we can decompose

$$a^p - 1 = (a^{\frac{p}{2}} + 1)(a^{\frac{p}{2}} - 1) = 0 \bmod N, \tag{9.17}$$

[12] in $\mathcal{O}((\log N)^3)$ time.

and therefore either one or both terms $(a^{\frac{p}{2}} \pm 1)$ must have a factor in common with N. Any non-trivial common divisor of N with $(a^{\frac{p}{2}} \pm 1)$, again calculated by Euclid's algorithm, yields thus a non-trivial factor of N.

Obviously, the described procedure is only successful if p is even and the final factor is a non-trivial one. Fortunately, if we choose a at random[13], this case occurs with probability larger than one half unless N is a power of a prime. The latter can, however, be checked again efficiently by a known classical algorithm, which returns the value of the prime. Altogether a polynomial time algorithm for determining the period of the function in Eq. (9.16) leads to a probabilistic polynomial time algorithm which either returns a factor of N or tells us that N is prime.

9.3.2 Quantum Fourier Transform

The step from the ordinary discrete Fourier transform (based on matrix multiplication) to the Fast Fourier Transform (FFT) has been of significant importance for signal and image processing as well as for many other applications in scientific and engineering computing[14]. Whereas the naive way of calculating the discrete Fourier transform

$$\hat{c}_y = \frac{1}{\sqrt{n}} \sum_{x=0}^{n-1} c_x e^{\frac{2\pi i}{n} xy} \tag{9.18}$$

by matrix multiplication takes $\mathcal{O}(n^2)$ steps, the FFT requires $\mathcal{O}(n \log n)$. The *quantum Fourier transform* (QFT) is in fact a straightforward quantum generalization of the FFT, which can, however, be implemented using only $\mathcal{O}((\log n)^2)$ elementary operations – an exponential speedup!

Let now the computational basis states of q qubits be characterized by the binary representation of numbers $x = \sum_{i=1}^{q} x_i 2^{i-1}$ via

$$|x\rangle = |x_1, \ldots, x_q\rangle. \tag{9.19}$$

That is, in this subsection x denotes from now on a natural number or zero and not a binary word. Then for $n = 2^q$ the QFT acts on a general state vector of q qubits as $\sum_x c_x |x\rangle \mapsto \sum_y \hat{c}_y |y\rangle$. This transformation can be implemented using only two types of gates: the Hadamard gate and conditional phase gates P_d acting as

$$|a, b\rangle \mapsto |a, b\rangle e^{\delta_{a+b,2} \pi i / 2^d}. \tag{9.20}$$

which rotate the relative phase conditionally by an angle $\pi 2^{-d}$, where d is the "distance" between the two involved qubits.

Fig. **??** shows the quantum circuit, which implements the QFT on $q = 3$ qubits. The extension of the circuit to more than three qubits is rather obvious and since $q(q+1)/2$ gates are required its complexity is $\mathcal{O}(q^2) = \mathcal{O}((\log n)^2)$. Being only interested in an

[13] For each randomly chosen a we have again to check whether $gcd(N, a) = 1$. The probability for this can be shown to be larger than $1/\log N$. The total probability of success is thus at least $1/(2 \log N)$.

[14] Although FFT is often attributed to Cooley and Tukey in 1965, it is now known that around 1805 Gauss used the algorithm already to interpolate the trajectories of asteroids.

approximate QFT we could reduce the number of gates even further to $\mathcal{O}(\log n)$ by dropping all phase gates P_d with $d \geq m$. Naturally, the accuracy will then depend on m.[15]

9.3.3 Joining the pieces

Let us now sketch how the QFT can be used to compute the period p of the function in Eq. (9.16) efficiently. Consider two registers of q qubits each, where $2^q = n \geq N^2$ and all the qubits are in the state vector $|0\rangle$ initially. Applying a Hadamard gate to each qubit in the first register yields $\sum_x |x, 0\rangle/\sqrt{n}$. Now suppose we have implemented the function in Eq. (9.16) in a quantum circuit which acts as $|x, 0\rangle \mapsto |x, f(x)\rangle$, where x is taken from $\mathbb{Z}_n$. Applying this to the state vector and then performing a QFT on the first register we obtain

$$\frac{1}{n} \sum_{x,y=0}^{n-1} e^{\frac{2\pi i}{n} xy} |y, f(x)\rangle. \tag{9.21}$$

How will the distribution of measurement outcomes look like if we now measure the first register in computational basis? Roughly speaking, the sum over x will lead to constructive interference whenever y/n is close to a multiple of the inverse of the period p of f and it yields destructive interference otherwise. Hence, the probability distribution for measuring y is sharply peaked around multiples of n/p and p itself can be determined by repeating the whole procedure $\mathcal{O}(\log N)$ times[16]. At the same time the probability of success can be made arbitrary close to one. In the end we can anyhow easily verify whether the result, the obtained factor of N, is valid or not.

What remains to be shown is that the map

$$|x, 0\rangle \mapsto |x, f(x)\rangle \ , \quad f(x) = a^x \bmod N \tag{9.22}$$

can be implemented efficiently. This can be done by repeatedly squaring in order to get $a^{2^j} \bmod N$ and then multiplying a subset of these numbers according to the binary expansion of x. This requires $\mathcal{O}(\log N)$ squarings and multiplications of $\log N$-bit numbers. For each multiplication the "elementary-school algorithm" requires $\mathcal{O}((\log N)^2)$ steps. Hence, implementing this simple classical algorithm on our quantum computer we can compute $f(x)$ with $\mathcal{O}((\log N)^3)$ elementary operations. In fact, this part of performing a standard classical multiplication algorithm on a quantum computer is the bottleneck in the quantum part of Shor's algorithm. If there would be a more refined *quantum modular exponentiation* algorithm we could improve the asymptotic performance of the algorithm[17].

[15] An ϵ-approximation of the QFT (in the 2-norm) would require $\mathcal{O}(q \log(q/\epsilon))$ operations, i.e., m is of the order $\log(q/\epsilon)$.

[16] For the cost of more classical postprocessing it is even possible to reduce the expected number of required trials to a constant.

[17] In fact, modular exponentiation can be done in $\mathcal{O}((\log N)^2 \log\log N \log\log\log N)$ time by utilizing the Schönhagen-Strassen algorithm for multiplication. However, this is again a classical algorithm, first made reversible and then run on a quantum computer. If there exists a faster quantum algorithm it would even be possible that breaking RSA codes on a quantum computer is asymptotically faster than the encryption on a classical computer.

Computational effort of Shor's algorithm: The quantum part of Shor's factoring algorithm requires of the order $(\log N)^3$ elementary steps

That is to say, the size of the circuit is cubic in the length of the input. As described above, additional classical preprocessing and post-processing is necessary in order to obtain a factor of N. The time required for the classical part of the algorithm is, however, polynomial in $\log N$ as well, such that the entire algorithm does the job in polynomial time.

Runtime of the best known algorithm for factoring: In contrast to that, the running time of the number field sieve, which is currently the best classical factoring algorithm, is $\exp[\mathcal{O}((\log N)^{\frac{1}{3}}(\log\log N)^{\frac{2}{3}})]$.

Moreover, it is widely believed that factoring is a classically hard problem, in the sense that there exists no classical polynomial time algorithm. However, it is also believed that proving the latter conjecture (if it is true) is extremely hard since it would solve the notorious $P \stackrel{?}{=} NP$ problem.

Shor's factoring algorithm falls into a certain class of quantum algorithms, together with many other important algorithms, such as the algorithm for computing orders of solvable groups and the efficient quantum algorithm for finding solutions of Pell's equation: it is an instance of a hidden subgroup problem. In fact, Shor's algorithm can be largely generalized to provide a solution of the *hidden sub-group problem*. Given a group G, a subgroup $H \leq G$, and a set X, we say a function $f : G \to X$ hides the subgroup$H$ if for all $g_1, g_2 \in G$,

$$f(g_1) = f(g_2) \tag{9.23}$$

holds true if and only if and only if $g_1 H = g_2 H$. Equivalently, the function f is constant on the cosets of H, while it is different between the different cosets of H. Again, this is a problem that can be efficiently solved on a quantum computer, while no polynomial time algorithm is known for the classical solution.

Hidden subgroup problem: Let G be a group, X a finite set, and $f : G \to X$ a function that hides a subgroup $H \leq G$. The function f is given via an oracle, which uses $\mathcal{O}(\log|G| + \log|X|)$ bits. Using information gained from evaluations of f via its oracle, determine a generating set for H.

9.4 Some thoughts on quantum algorithmic primitives

9.4.1 Quantum phase estimation

How can one find good quantum algorithms? There are no recipes. One has to use structure in a clever way. Each major quantum algorithm came along with a significant new idea. That said, important primitives play an important role, and are a part of many quantum algorithms. Among those primitives, it is the *quantum phase estimation* algorithm that is particularly important. The task is as follows.

Phase estimation problem: Consider a unitary U that acts on m qubits with an eigenvector $|\psi\rangle$ such that $U|\psi\rangle = e^{2\pi i\theta}|\psi\rangle$ for $0 \leq \theta < 1$. We would like to find the eigenvalue $e^{2\pi i\theta}$ of $|\psi\rangle$ which amounts to estimating the real phase θ, to a finite level of precision.

We $e^{2\pi i\theta}$ as the eigenvalues of a unitary all lie on the unit circle, and are hence complex numbers with absolute value 1. The quantum phase estimation algorithm is a subroutine of Shor's algorithm, but of also many others.

9.4.2 Other thoughts

There are many other primitives that one has made use of Algorithm based on *quantum random walks* constitute important examples of this kind, although the problem they solve do not appear in a particularly practical context. It has also been seen that the *exponentiation of structured matrices* such as sparse or low-rank matrices are a primitive that is often used. Using such a machinery, it has, e.g., in developing quantum algorithm that solve some *systems of linear equations* in logarithmic time, while the same problem takes polynomial time on a classical computer.