УДК 004.415.2, 004.588

QUANTUM ALGORITHMIC BENCHMARK'S GATE DESIGN AND SIMULATION OF QUANTUM SEARCH ALGORITHMS

Barchatova Irina[1], Fukuda Toshio[2], Ulyanov Sergey[3]

[1]PhD Student;
Dubna International University of Nature, Society and Man,
Institute of system analysis and management;
141980, Dubna, Moscow reg., Universitetskaya str., 19;
e-mail: i.a.barhatova@gmail.com.

[2]PhD, professor;
Dept. of Micro System, Dept. of Mechanics- Informatics, Nagoya University;
Furo-cho, Chikusa-ku, Nagoya, Japan;
e- mail: fukuda@mein.nagoya u.ac.jp.

[3]Doctor of Science in Physics and Mathematics, professor;
Dubna International University of Nature, Society and Man,
Institute of system analysis and management;
141980, Dubna, Moscow reg., Universitetskaya str., 19;
e-mail: ulyanovsv@mail.ru.

Main benchmark's gate design of quantum algorithms is introduced. Simulation results of quantum search algorithms on classical computers are described. Effective simulation methodology of quantum algorithms on classical computers are demonstrated.

Keywords: quantum search algorithms, simulation on classical computers, quantum algorithmic gate design.

ПРИМЕРЫ ПРОЕКТИРОВАНИЯ КВАНТОВЫХ АЛГОРИТМИЧЕСКИХ ЯЧЕЕК И МОДЕЛИРВОАНИЕ КВАНТОВЫХ ПОИСКОВЫХ АЛГОРИТМОВ

Бархатова Ирина Александровна[1], Фукуда Тошио[2], Ульянов Сергей Викторович[3]

[1]Аспирант;
ГБОУ ВО «Международный Университет природы, общества и человека «Дубна»,
Институт системного анализа и управления;
141980, Московская обл., г. Дубна, ул. Университетская, 19;
e-mail: i.a.barhatova@gmail.com.

[2]Доктор наук, профессор;
Факультет микросистем, механики и информатики;
Нагоя университет;
Япония, Нагоя, Фуро-чо;
e- mail: fukuda@mein.nagoya u.ac.jp.

[3]Доктор физико-математических наук, профессор;
ГБОУ ВО «Международный Университет природы, общества и человека «Дубна»,
Институт системного анализа и управления;
141980, Московская обл., г. Дубна, ул. Университетская, 19;
e-mail: ulyanovsv@mail.ru.

Рассмотрены основные примеры проектирования квантовых алгоритмических ячеек. Описываются результаты моделирования квантовых поисковых алгоритмов, реализованных на классическом компьютере. Продемонстрирована методология эффективного моделирования квантовых алгоритмов на классическом компьютере.

Ключевые слова: квантовые поисковые алгоритмы, моделирование на классическом компьютере, проектирование квантовых алгоритмических ячеек.

Simon's algorithm

Let us now examine Simon's algorithm. Unlike the Deutsch-Jozsa algorithm, Simon's algorithm has a nonzero error probability, i.e., it is not 100% certain to give the correct answer. However, the probability that it gives the correct answer is bigger than ½ if the conditions that we will state soon are satisfied, and the process of running the algorithm with such guarantee a number of times, and taking the value that shows up in a majority of the runs as your final output reduces the probability of ending up with a wrong result so dramatically that computer scientists are reasonably happy when they find fast probabilistic algorithms for their problems. (It does not make great sense to expect a fast zero-error run for almost any job from a quantum computer anyway, as we will see later.)

Simon's Problem: a black-box U_f which computes a function is given

$$f:\{0,1\}^n \to \{0,1\}^m \ (m \geq n)$$

that is known either to be one to one or to satisfy the equation

$$(x \neq y) \wedge \left(f(x) = f(y)\right) \leftrightarrow y = x \oplus s$$

for a non-trivial *s*, the problem is to determine if *f* is one to one, if it is not then to find *s*. (We denote the functionality of U_f as a unitary transformation: $U_f\left(|x\rangle \otimes |d\rangle\right) = |x\rangle \otimes \left|d \oplus f(x)\right\rangle$).

Algorithm: The «quantum part» of Simon's algorithm consists of the following steps:

Step 0: Initialize two quantum registers of n and m q-bits in the states

$$\bigotimes_0^{n-1}|0\rangle \ \textit{and} \ \bigotimes_0^{m-1}|0\rangle .$$

Step 1: Apply n-bit Hadamard gate H_n to the first register of n bits. The overall state will look as shown in the following equation.

Note that this step puts the first register into an equal superposition of the 2^n basis states

$$\left(H_n \bigotimes_0^{n-1}|0\rangle\right) \otimes \left(\bigotimes_0^{m-1}|0\rangle\right) = \left(\frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1}|k\rangle\right) \otimes \left(\bigotimes_0^{m-1}|0\rangle\right).$$

Step 2: Query the black-box for the state prepared in Step 1. Then the next state is

$$U_f\left(\left(\frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1}|k\rangle\right) \otimes \left(\bigotimes_0^{m-1}|0\rangle\right)\right) = \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1}|k\rangle \otimes \left|f(k)\right\rangle .$$

Step 3: Apply n-bit Hadamard gate H_n to the first register again. The new state is

$$(H_n \otimes I_m)\left(\frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1}|k\rangle \otimes \left|f(k)\right\rangle\right) = \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1}\left(\frac{1}{\sqrt{2^n}} \sum_{j=0}^{2^n-1}(-1)^{j \cdot k}|j\rangle \otimes \left|f(k)\right\rangle\right).$$

Now, if *f* is one to one then both the domain and range (mirror of the domain under *f*) of *f* has the same cardinality 2^n. The state shown above is a superposition of the members of the Cartesian product of the domain and range of *f*. Therefore, $2^n 2^n = 2^{2n}$ states of the form $|j\rangle \otimes \left|f(k)\right\rangle$ are superposed, each with an amplitude equal to either $\frac{1}{2^n}$ or $-\frac{1}{2^n}$. Then, if the state of the first register is measured after *Step 3*, the proba-

bilities for observing each of the 2^n basis states are equal and given by $2^n\left|\pm\frac{1}{2^n}\right|^2=\frac{1}{2^n}$. Hence the outcome of such a measurement would be a random value between 0 and 2^n-1.

If *f* is not one to one, then by the guarantee given to us about *s*, each state of the form $|j\rangle\otimes|f(k)\rangle$ has the amplitude given by

$$\frac{1}{2^n}\left((-1)^{j\cdot k}+(-1)^{j\cdot(k\oplus s)}\right)=\frac{1}{2^n}\left((-1)^{j\cdot k}+(-1)^{(j\cdot k)\oplus(j\cdot s)}\right).$$

If $j\cdot s\neq 0$ then the amplitude for $|j\rangle\otimes|f(k)\rangle$ becomes zero. So if *f* is not one to one, then measuring the state of the first register after *Step 3*, returns a value of *j* such that $j\cdot s=0$.

Step 4: Measure the state t of the first register.

We run these four steps $n-1$ times to form a system of equations of the form $t_1\cdot s=0, t_2\cdot s=0,..,t_{n-1}\cdot s=0$. If a nontrivial *s* exists, then these equations are linearly independent with probability at least ¼.

Why? Consider any $(i-1)$-member prefix of our sequence of observations, namely t_1, t_2, …, t_{i-1}, where $2 \le i \le n-1$. At most 2^{i-1} *n*-bit vectors are linear combinations of these. (Note that it is now useful to view the *t*'s as vectors of bits, and the relevant operation among them is ⊕.) Since the maximum number of *n*-bit vectors *t* such that $t\cdot s = 0$ is 2^{n-1}, the minimum number of vectors that are not the combinations of $i-1$ vectors is $2^{n-1}-2^{i-1}$. As a result, the probability that the i^{th} vector t_i is independent from the first $i-1$ vectors is at least $(2^{n-1}-2^{i-1})/2^{n-1}=1-\frac{1}{2^{n-i}}$. Using this fact, and the constraint that the first observation should not be the all-zero vector, we see that the probability that one obtains $n-1$ independent vectors is at least $\left(1-\frac{1}{2^{n-1}}\right)\left(1-\frac{1}{2^{n-2}}\right)\cdots\left(1-\frac{1}{2}\right)$ that can be shown to be greater than ¼ with a little bit of extra cleverness.

So, if the equations we obtained are really linearly independent, this system of *n*-1 linear equations in *n* unknowns can be solved for *n* bits of *s* classically, using Gaussian elimination, in polynomial time (since the unknowns are bits). If we get the value for *s*, we query the black-box twice to see if $f(0)=f(s)$. If this is the case, we conclude that *f* is two to one, and *s* has the value which has already been calculated. If not, (i.e. if we fail to solve the equations, or if the solution does not survive the black-box check,) we repeat the entire procedure. If we have not found an *s* which satisfies $f(0)=f(s)$ by the end of the third iteration of this outer loop, we claim that *f* is one to one, and stop.

If *f* is one to one, the algorithm says so with probability 1. Otherwise, it fails to find *n*-1 linearly independent equations in all three iterations and gives an incorrect answer only with probability at most $(3/4)^3 < (1/2)$. The runtime is clearly polynomial. The algorithm that solved this problem in exact polynomial time (with zero probability of error) has been also discovered, and just might incorporate it in a later release of these notes, so reloaded the problem once in a while.

The best classical probabilistic algorithm for the same task would require exponentially many queries of the black-box in terms of *n*. To see why, put yourself in place of such an algorithm. All you can do is to query the black-box with input after input and to hope to obtain some information about *s* from the outputs. (Let's assume that we are guaranteed that the function is two to one, and we are just looking for the string *s*.)

Note that, even if you don't get the same output to two different inputs, the outputs you get still say something about what *s* looks like, or rather, what it doesn't look like: If you have already made *k* queries without hitting the jackpot by receiving the same output to two different inputs, then you have learned that *s* is not one of the $\binom{k}{2}$ values that can be obtained by XORing any pair of the inputs that you have given. So next time you prepare an input, you will not consider any string which can be obtained by XORing one of those values by any previously entered input string. Even with all this sort of cleverness, the probability that

your next query will hit the jackpot is at most $\frac{k}{2^n - 1 - \binom{k}{2}}$, since, among the $2^n - 1 - \binom{k}{2}$ values for *s* that are still possible after your previous queries, (the −1 comes from the guarantee that *s* is nonzero) only *k* would let you solve the problem by examining the output of this query (by causing it to be the same as one of the previous *k* outputs, by being obtainable by XORing that previous input with the input to this query,) and, if the Universe is not trying to help or hinder you, the real *s* must be thought to be chosen uniformly at random from the set of all candidates. The probability of success after *m*+1 queries is not more than

$$\sum_{k=1}^{m} \frac{k}{2^n - 1 - \binom{k}{2}} \le \sum_{k=1}^{m} \frac{k}{2^n - k^2} \le \frac{m^2}{2^n - m^2}.$$

In order to be able to say that «this algorithm solves the problem with probability at least *p*», for a constant *p*, (*p* shouldn't decrease when *n* grows, as this makes the technique unusable for big *n*. If *p* is a nonzero constant, even a very small one, one can use the algorithm by running it repeatedly for only a constant number of times to find *s*.) We clearly have to set *m* to a value around at least $2^{n/2}$, which is exponential in terms of *n*.

Therefore, Simon's algorithm is exponentially faster than the best possible classical algorithm for the same task. Moreover, it is the first algorithm that depends on the idea of realizing the periodic properties of a function in the relative phase factors of a quantum state and then transforming it into information by means of probability distribution of the observed states. The ideas used in the period of finding algorithm turned out to be useful for developing algorithms for many other problems.

The Bershtein-Vazirani algorithm

Consider the binary function *f*(*x*) defined from an *n*-bit domain space to a 1-bit range $f : \{0,1\}^n \to \{0,1\}$. The function is considered to be of the form $f(x) = a \cdot x$, where *a* is an *n* bit string of zeros and ones and $a \cdot x$ denotes bitwise XOR (or scalar product modulo 2): $f(x) = a_1 \cdot x_1 \oplus a_2 \cdot x_2 \oplus \ldots \oplus a_n \cdot x_n$.

The aim of the algorithm is to find the *n*-bit string *a*, given that we have access to an oracle which gives us the values of the function *f*(*x*) when we supply it with an input *x*.

Classically at least *n* queries to the oracle are required in order to find the binary string *a*. The Bernstein-Vazirani algorithm solves this problem with a single query to a quantum oracle of the form

$$|x\rangle_{n-\text{qubit}} |y\rangle_{1-\text{qubit}} \xrightarrow{U_a} |x\rangle_{n-\text{qubit}} |f(x) \oplus y\rangle_{1-\text{qubit}} \quad (1)$$

where $x \in \{0,\ldots,2^{n-1}\}$ is a data register and $|y\rangle$ acts as a target register.

The algorithm works as follows: begin with an initial state with the first *n*-q-bits in $|0\rangle$ state and the last q-bit in the state $|1\rangle$. Apply a Hadamard transformation on all the *n* + 1 q-bits and then make a call to the oracle giving the following results:

$$\begin{aligned} |0^n\rangle|1\rangle &\xrightarrow{H^{\otimes n+1}} \frac{1}{\sqrt[n]{2}} \sum_{x=0}^{2^n-1} |x\rangle \frac{1}{\sqrt{2}} \left(|0\rangle - |1\rangle\right) \xrightarrow{U_a} \frac{1}{\sqrt[n]{2}} \sum_{x=0}^{2^n-1} \left(-1^{(x \cdot a)}\right) |x\rangle \frac{1}{\sqrt{2}} \left(|0\rangle - |1\rangle\right) \\ &\xrightarrow{H^{\otimes n+1}} \frac{1}{2^n} \sum_{x=0}^{2^n-1} \sum_{z=0}^{2^n-1} \left(-1^{(x \cdot a)}\right) \left(-1^{(x \cdot z)}\right) |z\rangle|1\rangle \equiv |a\rangle|1\rangle_{az} \end{aligned} \quad (2)$$

where we have used the fact that $\frac{1}{2^n} \sum_{x=0}^{2^n-1} \sum_{z=0}^{2^n-1} \left(-1^{(x \cdot a)}\right) \left(-1^{(x \cdot z)}\right) = \delta_{az}$.

A measurement in the computational basis immediately reveals the binary vector *a*. This algorithm therefore achieves the discovery of the vector *a* in a single oracle call as opposed to *n* oracle calls required classically. The oracle (1.5) has been queried on a superposition of states for this algorithm. However, if we query the oracle on a state $|x\rangle$ with the function register set to $|0\rangle$ we will recover the value *f*(*x*) in the function register. This demonstrates that we can run the oracle in the classical mode when desired.

Oracle modification and implementation without entanglement. This unitary oracle (1.5) requires *n* + 1 q-bits and can be operated in two different ways. If we use eigenstates in the input and set *y* = 0, the algorithm outputs *f*(*x*) for a given input x in a reversible manner (which the original classical algorithm would do irreversibly). However, the algorithm can be performed on arbitrary quantum states (typically a uniform superposition of input states in the Deutsch-Jozsa and Bernstein-Vazirani algorithms).

A careful perusal of Eq. (2) reveals two important facts about the Bernstein-Vazirani algorithm.

(a) The register q-bit does not play any role in the algorithm. It is used only in the function evaluation step because the oracle (1) demands that we supply this extra q-bit. However, if we modify the oracle to

$$|x\rangle_{n-\text{bit}} \xrightarrow{U_f} (-1)^{f(x)} |x\rangle_{n-\text{bit}} \tag{3}$$

we can implement everything on *n* q-bits. Since the state of the last q-bit does not change, it can be considered redundant and we can remove the one-q-bit target register altogether.

Remark. Although this oracle suffices to execute the Bernstein-Vazirani algorithm, it cannot give us the value of *f*(*x*) for a given *x*. Therefore, one can argue that the connection with the original classical problem is lost and one is solving altogether different problem. We demonstrate that in the classical model based on polarization of light beams, this problem can be circumvented and we can obtain the value of *f*(*x*) from *x* via a suitable modification of the circuit. We will come back to these subtle points again in the next section.

(b) It turns out that this version of the oracle is implementable without requiring any entanglement for the case of the Bernstein-Vazirani algorithm. The modified oracle (1.7) can be implemented without introducing any entanglement because the unitary transformation U_a can be decomposed as a direct product of single q-bit operations

$$U_a = U_a^{(1)} \otimes U_a^{(2)} \otimes \cdots U_a^{(n)} = \left(\sigma_z^1\right)^{a_1} \otimes \left(\sigma_z^2\right)^{a_2} \otimes \cdots \otimes \left(\sigma_z^n\right)^{a_n},$$

where σ_z^j is the Pauli operator acting on the *j* th q-bit. On an *n*-q-bit eigenstate $|x\rangle = |x_1\rangle|x_2\rangle\cdots|x_n\rangle$ labeled by the binary string x the action reduces to

$$U_a \equiv (-1)^{x_1 \cdot a_1} (-1)^{x_2 \cdot a_2} \cdots (-1)^{x_n \cdot a_n}.$$

This simplified version of the Bernstein-Vazirani algorithm where only *n* q-bits are used and we have separable states at all stages of the is depicted in Fig. 1.

Initially the q-bits are set to be all in the $|0\rangle$ state. Each box containing *H* represents a Hadamard transformation and the box U_a represents the oracle. By a single call to the oracle sandwiched between the Hadamard gates we arrive at the final state $|a\rangle$ which reveals the binary vector *a* on measurement.

All the implementations till date have been along the lines of this circuit.

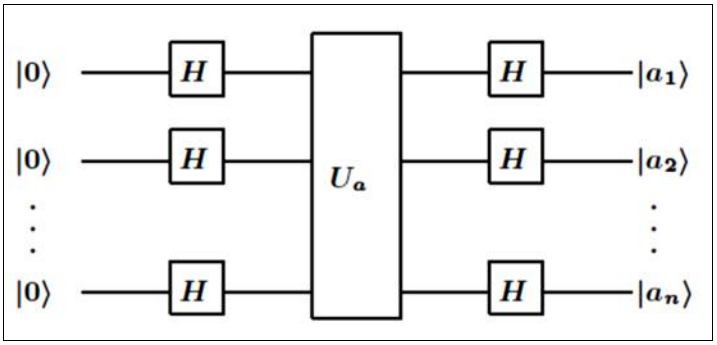


Figure 1. Representation of the Bernstein-Vazirani algorithm using a modified oracle on n unentangled q-bits

Grover's quantum search algorithm

We will now examine Grover's algorithm for function inversion that can be used, as example, to search a phone book with *N* entries to find the name corresponding to a given phone number in $O(\sqrt{N})$ steps (the best classical algorithms can do this in $O(N)$ steps).

Simple introduction

Assume that we are given a quantum oracle ***G*** for computing the Boolean function *f* which is known to return 1 for exactly one possible input number, and 0 for all the remaining $2^n - 1$ possible inputs. As in our previous examples, the oracle operates on $n+1$ q-bits, such that the input $|x\rangle \otimes |d\rangle$ is transformed to the output $|x\rangle \otimes |d \oplus f(x)\rangle$.

Our task is to find which particular value of *x* makes $f(x) = 1$.

Here is Grover's algorithm for an $n + 1$ q-bit register, where $n > 2$:

1. Initialize the register so that the first *n* q-bits are $|0\rangle$, and the last one is $|1\rangle$.
2. Apply the *H* gate to each q-bit.
3. Do the following *r* times, where *r* is the nearest integer to $\dfrac{\cos^{-1}\left(\frac{1}{\sqrt{2^n}}\right)}{2\sin^{-1}\left(\frac{1}{\sqrt{2^n}}\right)}$:

   (a) Apply ***G*** to the register.

   (b) Apply the program ***V*** that will be described below, to the first *n* q-bits.
4. Read (measure) the number written in the first *n* q-bits, and claim that this is the value that makes *f* 1.

The program ***V*** is defined as the 2^n by 2^n matrix that has the number $\dfrac{1-2^{n-1}}{2^{n-1}}$ in all its main diagonal entries and $\dfrac{1}{2^{n-1}}$ everywhere else.

Let us see why the algorithm works correctly:

At the end of stage 2 the state of the register is $\dfrac{1}{\sqrt{2^n}}\sum_{x=0}^{2^n-1}|x\rangle \otimes \left[\dfrac{|0\rangle - |1\rangle}{\sqrt{2}}\right]$. We will now examine what each iteration of the loop does to the register.

At the end of the first execution of stage 3.a we have such a term of the state in the register for each *x* in the summation:

$$\frac{1}{\sqrt{2^n}}|x\rangle\otimes\left[\frac{\left(|0\rangle\oplus f(x)\right)-\left(|1\rangle\oplus f(x)\right)}{\sqrt{2}}\right]=\begin{cases}\frac{1}{\sqrt{2^n}}|x\rangle\otimes\left[\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right] & \text{if } f(x)=0\\ \frac{1}{\sqrt{2^n}}|x\rangle\otimes\left[\frac{|1\rangle-|0\rangle}{\sqrt{2}}\right] & \text{if } f(x)=1\end{cases}$$

$$=\frac{1}{\sqrt{2^n}}|x\rangle\otimes(-1)^{f(x)}\left[\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right]$$

$$=\frac{1}{\sqrt{2^n}}(-1)^{f(x)}|x\rangle\otimes\left[\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right].$$

Now let us generalize this to an arbitrary execution of this stage where the amplitude of each $|x\rangle$ need not be equal. It is easy to see that if the last q-bit is $\frac{1}{\sqrt{2}}|0\rangle-\frac{1}{\sqrt{2}}|1\rangle$ to start with the resulting amplitude of a particular $|x\rangle$ remains the same if and only if $f(x) = 0$. For the one $|x\rangle$ value which makes $f(x) = 1$ the sign of the amplitude is reversed. And the last q-bit remains unchanged at $\frac{1}{\sqrt{2}}|0\rangle-\frac{1}{\sqrt{2}}|1\rangle$. We can therefore "forget" the last q-bit and view stage 3.a as the application of an *n*-q-bit program which flips the sign of the amplitude of the searched number $|w\rangle$ and leaves everything else unchanged in the first *n* q-bits.

Now let's get to stage 3b. The best way of understanding the program ***V*** is to see it as the sum of the 2^n by 2^n matrix which has the number $\frac{1}{2^{n-1}}$ in all its entries and the 2^n by 2^n matrix which has -1 in all its main diagonal entries and 0 everywhere else. Using the properties of matrix multiplication and addition we can analyze the effect of this stage on our *n* q-bits by analyzing the results that would be obtained if these two matrices are applied separately on the q-bits and then adding the two result vectors. Say that our *n*-q-bit collection has the following state before the execution of this stage: $\begin{pmatrix}\alpha_0\\ \vdots\\ \alpha_{2^n-1}\end{pmatrix}$. Multiplication with the 2^n by 2^n matrix which has the number $\frac{1}{2^{n-1}}$ in all its entries yields the state $\begin{pmatrix}2a\\ \vdots\\ 2a\end{pmatrix}$ where *a* is the average of the amplitudes α_i in the original state.

On the other hand, multiplying $\begin{pmatrix}\alpha_0\\ \vdots\\ \alpha_{2^n-1}\end{pmatrix}$ with the 2^n by 2^n matrix which has -1 in all its main diagonal entries and 0 everywhere else just flips the signs of all the α_i in the original state.

So what do we get when we add these two result vectors? Each basis state $|x\rangle$ which had the amplitude α_x before stage 3.b obtains the amplitude $2a-\alpha_x$ at the end of this stage. A useful way of interpreting this stage is saying that each amplitude α_x is mapped to $a+(a-\alpha_x)$, i.e. every amplitude that was *u* units above (below) the average amplitude before the execution would be at *u* units below (above) the average amplitude after the execution.

At this point we start seeing the logic of Grover's algorithm: In the beginning all the 2^n numbers in the *n*-q-bit collection have the same amplitude, namely $\frac{1}{\sqrt{2^n}}$. Stage 3.a flips the sign of the amplitude of the searched number $|w\rangle$, so now the amplitudes of all the other numbers are very close to the average amplitude *a* but the amplitude of $|w\rangle$ is approximately –*a* at a distance of nearly 2*a* from the average. Stage 3.b restores the sign of $|w\rangle$'s amplitude to positive but during this process its value becomes approximately 3*a*. It is clear that the amplitude of $|w\rangle$ and therefore the probability that $|w\rangle$ will be observed when we read the first *n* q-bits will grow further and further as the iterations of the loop continue and this amplitude will reach a value greater than $\frac{1}{\sqrt{2}}$ after a certain number of iterations.

Note that if the amplitude of $|w\rangle$ grows so much that the average at the beginning of stage 3.b becomes negative each iteration would start *shrinking* rather than growing the amplitude of $|w\rangle$, so it is important to know exactly how many times this loop should be iterated.

We are now supposed to find what that required number of iterations that is also essential in the calculation of the time complexity of this algorithm is. To make this calculation let us adopt a geometric visualization of the vectors we use to represent the states of our quantum registers. If we are talking about an *n*-q-bit collection as in this case there are 2^n basis states as it is already mentioned. An arbitrary state of our collection is then a vector with length one in the 2^n-dimensional space where each of those basis states can be viewed to be at an angle of π/2 radians from each other. An arbitrary state $\alpha_0|0\rangle+\alpha_1|1\rangle+\cdots+\alpha_{2^{n-1}}|2^{n-1}\rangle$ is just the vector sum of all the 2^n component vectors in the expression for it. The length of each of these component vectors is just the absolute value of the corresponding amplitude.

The application of a quantum program on an *n*-q-bit register has just the effect of rotating the unit vector representing its state to a new alignment in this space. The probability of observation of a particular basis value *x* when we are at an arbitrary state $|\psi\rangle=\alpha_0|0\rangle+\alpha_1|1\rangle+\cdots+\alpha_{2^{n-1}}|2^{n-1}\rangle$ can be viewed as the square of the length of the projection of vector $|\psi\rangle$ onto the vector $|x\rangle$. This projection's length is just equal to the cosine of the angle between $|\psi\rangle$ and $|x\rangle$. (Do not worry about the amplitudes being complex numbers in general; most what you already know about vectors is still valid here. By the way, in the particular example of Grover's algorithm the amplitudes have no imaginary components at any stage of the execution.)

With this visualization method in mind we will examine what a single iteration of stage 3 makes to the *n*-q-bit state. We already know that stage 3.a flips the sign of the amplitude of the component vector corresponding to searched number $|w\rangle$ and leaves everything else unchanged.

In the beginning of the first iteration, the state vector is $|s\rangle=\frac{1}{\sqrt{2^n}}\sum_{x=0}^{2^n-1}|x\rangle$. So the angle between vector $|s\rangle$ and each one of the basis state vectors is $\cos^{-1}\left(\frac{1}{\sqrt{2^n}}\right)$. In particular the angle between $|s\rangle$ and $|w\rangle$ is $\cos^{-1}\left(\frac{1}{\sqrt{2^n}}\right)$. Think about the initial vector as the sum $\alpha|w\rangle+\beta|y\rangle$ where the second term is just what you get if you subtract $\alpha|w\rangle$ from $\frac{1}{\sqrt{2^n}}\sum_{x=0}^{2^n-1}|x\rangle$ and $|y\rangle$ is the unit vector in that direction.

Since the job of stage 3.a is to flip the sign of the amplitude of $|w\rangle$ and leave everything else unchanged the resulting vector is $-\alpha|w\rangle+\beta|y\rangle$. The angle between this new vector and $|w\rangle$ is $\cos^{-1}\left(\frac{-1}{\sqrt{2^n}}\right)$, i.e. the

same as before. The vector rotated all right but it ended up at the same angle from all the basis state vectors. Only its alignment changed with respect to $|w\rangle$.

It is important to see that this is what *any* iteration of stage 3.a does: *reflects* the current state vector around the $|y\rangle$ axis in the plane defined by the $|w\rangle$ and $|y\rangle$ axes. To have a deeper understanding of what is going on let's introduce more of the notation used in the quantum literature. For any column vector $|v\rangle$ the expression $\langle v|$ denotes the row vector obtained by replacing each component $a+b\cdot i$ of $|v\rangle$ with the number $a-b\cdot i$ and then writing these in a row. Note that the expression $|v\rangle\langle v|$ describes a square matrix. Now the matrix of stage 3.a can be seen to equal $I-2|w\rangle\langle w|$ where *I* is the identity matrix of the appropriate size.

So we know that a program of the form $I-2|w\rangle\langle w|$ when applied to a register in state $\alpha|w\rangle+\beta|y\rangle$ where the angle between the $|w\rangle$ and $|y\rangle$ vectors is π/2 reflects the state vector around the $|y\rangle$ axis in the plane defined by the $|w\rangle$ and $|y\rangle$ axes. But here comes another surprise: It turns out that in quantum computing multiplying a program's matrix with any number of the form e^{ix} where *x* is any real number (recall that $e^{ix}=\cos x+i\sin x$) yields a program which is completely equivalent to the original one from the point of view of the user. (This is because the measurement probabilities corresponding to the two amplitudes $a+b\cdot i$ and $e^{ix}(a+b\cdot i)$ are identical as you can check using the formula for e^{ix} given above.) This means that $I-2|w\rangle\langle w|$ and $2|w\rangle\langle w|-I$ (which is just $I-2|w\rangle\langle w|$ multiplied by $-1=e^{i\pi}$) can be interchanged without changing the functionality of the program.

So we can replace the program of stage 3.a with the completely equivalent program $2|w\rangle\langle w|-I$ that can be easily seen to transform the input $\alpha|w\rangle+\beta|y\rangle$ to $\alpha|w\rangle-\beta|y\rangle$, i.e. to reflect it around the $|w\rangle$ axis if we prefer such an interpretation.

Now consider the program of stage 3.b: It is easy to see that this program equals $2|s\rangle\langle s|-I$ where $|s\rangle=\frac{1}{\sqrt{2^n}}\sum_{x=0}^{2^n-1}|x\rangle$ is the state of the *n*-q-bit register at the end of stage 2. According to our earlier discussion $2|s\rangle\langle s|-I$ and $I-2|s\rangle\langle s|$ can be interchanged and we can say that stage 3.b just reflects its input vector around the $|s\rangle$ axis.

Now think about the plane defined by the $|w\rangle$ and $|y\rangle$ axes with $|y\rangle$ horizontal and $|w\rangle$ vertical. After stage 2 our state is $|s\rangle=\frac{1}{\sqrt{2^n}}|w\rangle+\sqrt{\frac{2^n-1}{2^n}}|y\rangle$, so it is a vector within the first quadrant of this plane. The angle between $|s\rangle$ and $|y\rangle$ can be seen as $\sin^{-1}\left(\frac{1}{\sqrt{2^n}}\right)$.

When stage 3.a acts our vector is reflected around the $|y\rangle$ axis. Since $|s\rangle$ itself was in the $|w\rangle$-$|y\rangle$ plane and we reflected it around $|y\rangle$ the resulting vector is still in the $|w\rangle$-$|y\rangle$ plane, $\sin^{-1}\left(\frac{1}{\sqrt{2^n}}\right)$ radians away from the $|y\rangle$ axis in the fourth quadrant.

When stage 3.b acts this vector is reflected around the $|s\rangle$ axis. The resulting vector is in the $|w\rangle$-$|y\rangle$ plane, and it is now $3.\sin^{-1}\left(\frac{1}{\sqrt{2^n}}\right)$ radians from the $|y\rangle$ axis in the first quadrant once again. It is easy to see that the combined effect of stage 3 for any iteration in this algorithm is to rotate the vector that it finds for

$2.\sin^{-1}\left(\frac{1}{\sqrt{2^n}}\right)$ radians in the counterclockwise direction in the $|w\rangle$-$|y\rangle$ plane. We want to iterate the loop until the state gets sufficiently close to the $|w\rangle$ axis so that the probability of observing $|w\rangle$ during a measurement is greater than ½.

Recall that the state vector was $\cos^{-1}\left(\frac{1}{\sqrt{2^n}}\right)$ radians away from the $|w\rangle$ axis before the loop. It is clear that a number *r* of iterations where

$$r\left(2\sin^{-1}\left(\frac{1}{\sqrt{2^n}}\right)\right)=\cos^{-1}\left(\frac{1}{\sqrt{2^n}}\right)$$

would be ideal. Since *r* has to be an integer this is not always possible so we settle for

$$r = round\left(\frac{\cos^{-1}\left(\frac{1}{\sqrt{2^n}}\right)}{2\sin^{-1}\left(\frac{1}{\sqrt{2^n}}\right)}\right).$$

Note that if the loop iterates this many times the resulting vector can be at most $\sin^{-1}\left(\frac{1}{\sqrt{2^n}}\right)$ radians away from the $|w\rangle$ axis, and the probability of observing $|w\rangle$ at stage 4 would be at least $\cos^2\left(\sin^{-1}\left(\frac{1}{\sqrt{2^n}}\right)\right)$ what is greater than ½ for all $n>2$.

So what is the time complexity of Grover's algorithm? If we measure it in terms of $N=2^n$ which is the size of the "database" being searched the loop iterates $round\left(\frac{\cos^{-1}\left(\frac{1}{\sqrt{N}}\right)}{2\sin^{-1}\left(\frac{1}{\sqrt{N}}\right)}\right)$ times.

For big *N* the numerator approaches $\frac{\pi}{2}$ whereas $\sin^{-1}$ tends to near the value of its own argument when the argument is nearing zero, so we can see that the time complexity is indeed O($\sqrt{N}$). The reasoning mentioned above can be generalized easily to the case where the number of inputs for which *f* returns 1 is not one but $M \le N/2$.

Note that in all the algorithms stated above we analyzed only the number of required quantum oracle calls and showed that they were better than the number of required classical oracle calls in the best possible classical algorithms for that job. A complete analysis would also require quantifying the amount of resources (for instance, in terms of the number of elementary quantum gates from a fixed set) that would be required to implement the rest of the algorithms as the function of the size of the input. Although we do not give that analysis here those algorithms turn out to have good (polynomial) complexities in that regard as well.

Grover's QSA: models of quantum oracles and computational algorithm

Grover's search algorithm provides an example of the speed-up that would be offered by quantum computers (if and when they are built) and has the important application in solution of *global optimization* control problems. The problem solved by Grover's algorithm is finding a sought-after («*marked*») element in an unsorted database (DB) of size *N* .

To solve this problem a classical computer would need $\frac{N}{2}$ database queries on average and in the worst case it would do $N-1$ queries. Using Grover's algorithm a quantum computer can find the marked state using only $O\left(\sqrt{N}\right)$ quantum data queries. In the case of M «*marked*» elements in an unsorted DB of size N speed-up of quantum search process increases as $O\left(\sqrt{\frac{N}{M}}\right)$.

Remark. Related works and optimality of quantum searching Grover discovered a QA for identifying a target element in an unstructured DB search universe of N items in approximately $\frac{\pi}{4}\sqrt{N}$ queries to a quantum oracle. For classical search using a classical oracle the search complexity is clearly of order $\frac{N}{2}$ queries since half of the items must be searched on average. It has been proved that this square-root speeds-up the best attainable performance gained by any QA. Its work preceding Grover's, *Bennett et al*. (1997) showed that no QA could solve the search problem in fewer than $O\left(\sqrt{N}\right)$ queries. Following Grover's work, *Boyer et al*. (1998) showed that Grover's algorithm was optimal asymptotically and that square-root speed-up cannot be improved even if one allows e.g., a 50% probability of error. *Zalka* (1999) strengthened these results to show that Grover's algorithm is an optimal algorithm exactly (not only asymptotically). In this correspondence we will also present an *information-theoretic* analysis of Grover's algorithm and discuss the optimality of Grover's algorithm from a different point of view for application in design of robust intelligent control.

Thus, Grover's algorithm has optimal order of complexity. Here we present an information-theoretic analysis of Grover's algorithm and show that the square-root speed-up by Grover's algorithm is the best possible by any algorithm using the same quantum oracle.

Search problem for an unstructured DB

Consider the problem of searching an unstructured DB of $N=2^n$ records for exactly one record that has been specifically marked. This can be rephrased in mathematical terms as *an oracle problem* as follows. Label the records of the DB with the integers $0,1,2,...,N-1$ and denote the label of unknown marked record by x_0. We are given *an oracle* which computes the *n*-bit binary function $f:\{0,1\}^n \to \{0,1\}$ defined by $f(x)=\begin{cases}1, & \text{if } x=x_0\\ 0, & \text{otherwise.}\end{cases}$.

As a standard oracle idealization we have no access to the internal workings of the function *f*. It operates simply as a black-box function, that we can query as many as we like. But with each such ~~a~~ query an associated computational cost comes.

Search problem for an unstructured DB: Find the record labeled as x_0 with the minimum amount of computational work, i.e., with the minimum number of queries of the oracle f.

Remark. From the probability theory we know that if we examine *k* records, i.e. if we compute the oracle *f* for *k* randomly chosen records then the probability of finding the record is labeled as x_0 is $\frac{k}{N}$. Hence, on a classical computer it takes $O(N)=O(2^n)$ queries to find the record labeled x_0. However, as Grover so astutely observed on a quantum computer the search of an unstructured database can be accomplish in $O(\sqrt{N})$ steps or more precisely with the application of $O(\sqrt{N}\lg N)$ sufficiently local unitary transformations. Although this is not exponentially faster it is a significant speed-up.

Main steps of Grover's QSA

We assume without loss of generality that $N = 2^n$ where n is an integer. The algorithm requires of n q-bits carrying the computation. When we say it is in a state $|x\rangle$ we mean that its q-bits are in states corresponding to the binary representation of the number x.

Example. Consider the following *Problem*:

Input	$x_1 \in \{0,1\}, x_2 \in \{0,1\}, \ldots, x_N \in \{0,1\}$ such that exactly one x_i is 1.
Output	The i such that $x_i = 1$.

Classically, one needs $\Omega(N)$ queries to solve this problem and there is no better algorithm than the locations one by one until we find $x_i = 1$. Surprisingly, there is a better algorithm in the quantum case (*Grover*, 1996): *There is a QA that solves Problem with* $O(\sqrt{N})$ *queries.*

Qualitatively Grover's original QSA consists of the following *steps*:

(1). Initialize the register to $H|0\rangle$. Reset all q-bits to 0 and apply the Hadamard transform to each of them;

(2). Repeat the following operation (named the *Grover iterate G*) $T = \frac{\pi}{4}\sqrt{N}$ times:

(2.a) Rotate the marked state $|k\rangle$ by a phase of π radians (I_k^π);

(2.b) Apply the Hadamard transform to the register;

(2.c) Rotate the $|0\rangle$ state by a phase of π radians (I_0^π);

(2.d) Apply the Hadamard transform again;

(3). Measure the resulting state.

Remark. The original Grover's iterate is $Q = -HI_0^\pi HI_k^\pi$. It has been generalized to $Q = -UI_s^\beta U^\dagger I_M^\gamma$ where U is an arbitrary unitary operator, s is an arbitrary state, variables β and γ are arbitrary angles and M includes any number of marked states. We have now observed that any unitary operation Q has a unitary diagonalization. Therefore, it can be represented as $Q = -UI_{\vec{S}}^{\vec{\beta}} U^\dagger I_M^\gamma$. This is a further generalization of Grover's algorithm where the state s is replaced by a set of states $\vec{S}$, each of which may have a different rotation angle. Thus, every iterative algorithm is a generalized Grover's algorithm.

According to abovementioned QSA in computation steps of this we shall:

(4). Apply a unitary transformation U mapping $|0\rangle$ to $\frac{1}{\sqrt{N}}\sum_{i=0}^{N-1}|i\rangle$;

(5). Repeat for $\left\lceil \frac{\pi}{4}\sqrt{N} \right\rceil$ times:

- Apply the query transformation O which maps $\sum_{i=0}^{N-1} a_i |i\rangle$ to

$$\sum_{i=0}^{N-1} a_i (-1)^{x_i} |i\rangle ;$$

- Apply the following «diffusion operator D»

$$\begin{cases} D|1\rangle = -\dfrac{N-2}{N}|1\rangle + \dfrac{2}{N}|2\rangle + \ldots + \dfrac{2}{N}|N\rangle \\ D|2\rangle = \dfrac{2}{N}|1\rangle - \dfrac{N-2}{N}|2\rangle + \ldots + \dfrac{2}{N}|N\rangle \\ \ldots \\ D|N\rangle = \dfrac{2}{N}|1\rangle + \dfrac{2}{N}|2\rangle + \ldots - \dfrac{N-2}{N}|N\rangle \end{cases}$$

(6). Measure the state and output the result of the measurement.

Remark. We note that Grover's QSA is efficient not just in the number of queries but also in the running time. The reason for that is that the diffusion operator D can be implemented in $O(logN)$ time steps. Therefore, the whole algorithm can be implemented in $O(\sqrt{N}\, logN)$.

Example. Mathematical properties of quantum operations in QSA. Let $\mathcal{H}_2$ be a 2 dimensional Hilbert space with orthonormal basis $\{|0\rangle,|1\rangle\}$ and let the set $\{|0\rangle,|1\rangle,...,|N-1\rangle\}$ denote the induced orthonormal basis in the Hilbert space $\mathcal{H} = \bigotimes_0^{N-1} \mathcal{H}_2$. From the quantum mechanical perspective the oracle function *f* is given as a black-box transformation U_f, i.e., by

$\mathcal{H} \otimes \mathcal{H}_2 \overset{U_f}{\rightarrow} \mathcal{H} \otimes \mathcal{H}_2$
$\lvert x\rangle \otimes \lvert y\rangle \overset{U_f}{\rightarrow} \lvert x\rangle \otimes \lvert f(x) \oplus y\rangle,$
where « $\oplus$ » denotes exclusive *OR* – XOR, i.e. addition modulo 2.

Remark. Instead of U_f we will use below the computationally equivalent unitary transformation

$$I_{|x_0\rangle}(|x\rangle) = (-1)^{f(x)}|x\rangle = \begin{cases} -|x_0\rangle, & \text{if } x = x_0 \\ |x\rangle, & \text{otherwise} \end{cases}$$

That $I_{|x_0\rangle}$ is computationally equivalent to U_f follows from the easily verifiable fact that

$$U_f\left(|x\rangle \otimes \frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) = \left(I_{|x_0\rangle}(|x\rangle)\right) \otimes \frac{|0\rangle - |1\rangle}{\sqrt{2}}$$

and also from the fact that U_f can be constructed from a controlled $I_{|x_0\rangle}$ and two one q-bit Hadamard transforms.

We now informally explain why this QSA work follows the «*inversion against average*» method. To understand the algorithm plot the amplitudes of $|1\rangle,\ldots,|N\rangle$ at each step. After the first step the state is $\frac{1}{\sqrt{N}}\sum_{i=0}^{N-1}|i\rangle$ and the amplitudes are $\frac{1}{\sqrt{N}}$.

Fig. 2 (a) shows this result.

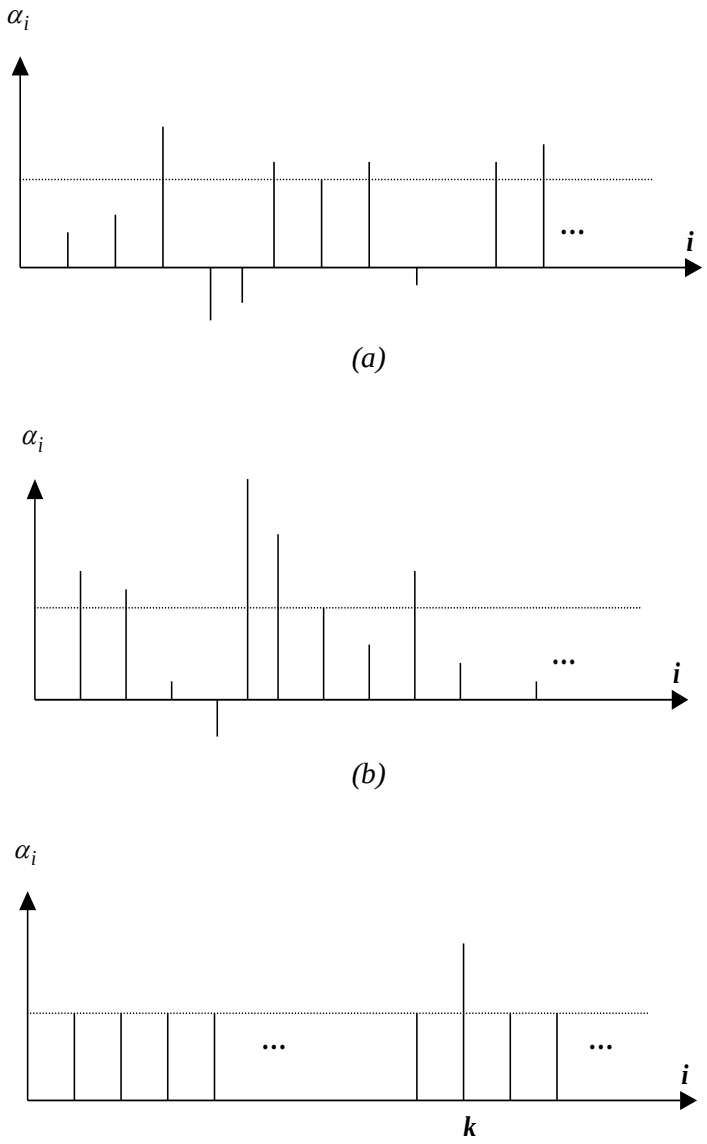


Figure 2. Effects of D operation: (a) States before operation; (b) States after operation; (c) result of calculations

After the first query the amplitude of $|i\rangle$ with $x_i = 1$ becomes $\left(-\frac{1}{\sqrt{N}}\right)$. Fig. 2 (b) shows this result. Then the diffusion operator D is applied. Let $|\psi\rangle = \sum_{i=0}^{N-1} a_i |i\rangle$ be the state before the action of D. Then the state after the action of D is $|\psi'\rangle = \sum_{i=0}^{N-1} a_i' |i\rangle$ where $a_i' = -\frac{N-2}{N} a_i + \sum_{i \neq j} \frac{2}{N} a_j$. We can rewrite this as $a_i' = -a_i + \sum_{j=1}^{N} \frac{2}{N} a_j$ and $a_i' + a_i = \sum_{j=1}^{N} \frac{2}{N} a_j$. Let $A = \sum_{j=1}^{N} \frac{1}{N} a_j$ be the average of probability amplitudes a_i. Thus, we have $a_i' + a_i = 2A$ and if $a_i = A + \Delta$ then $a_i' = A - \Delta$. Therefore, the effect of the «diffusion transform» is that every amplitude a_i is replaced by its reflection against the average of all a_i.

In particular, after the first query the amplitude of $|i\rangle$ with $x_i = 1$ is $\left(-\frac{1}{\sqrt{N}}\right)$ and all the other amplitudes are $\left(\frac{1}{\sqrt{N}}\right)$. The average is $\left(\frac{1}{\sqrt{N}} - \frac{2}{N\sqrt{N}}\right)$ that is almost $\frac{1}{\sqrt{N}}$.

Therefore, after applying D the amplitude of $|i\rangle$ with $x_i = 1$ becomes almost $\frac{3}{\sqrt{N}}$ and the amplitudes of all other basis states $|j\rangle$ do slightly less than $\frac{1}{\sqrt{N}}$.

Fig. 2 (c) shows this result of calculations. The next query makes the amplitude of $|i\rangle$ with $x_i = 1$ approximately $\left(-\frac{3}{\sqrt{N}}\right)$.

The average of all amplitudes is slightly less than $\frac{1}{\sqrt{N}}$ and reflecting against it makes the amplitudes of $|i\rangle$ with $x_i = 1$ approximately $\left(\frac{5}{\sqrt{N}}\right)$.

Thus, each step increases the amplitude of $|i\rangle$ with $x_i = 1$ by $O\left(\frac{1}{\sqrt{N}}\right)$ and decreases the other amplitudes. A precise calculation shows that after $\frac{\pi}{4}\sqrt{N}$ steps the amplitude of $|i\rangle$ with $x_i = 1$ is $1 - o(1)$. Therefore, the measurement gives the correct answer with probability of $1 - o(1)$.

Remark. Boyer et all (1998) have extended Grover's QSA to the case when there can be more than one i with $x_i = 1$. The simplest case if the number of $x_i = 1$ is known. If there are k such values we can run the same algorithm with $\left\lceil \frac{\pi}{4}\sqrt{\frac{N}{k}} \right\rceil$ iterations instead of $\left\lceil \frac{\pi}{4}\sqrt{N} \right\rceil$. An analysis similar to one above shows that this gives a random i such that $x_i = 1$ with high probability.

Example. A more difficult case is if k is not known in advance. The problem is that after reaching the maximum the amplitudes of i with $x_i = 1$ start to decrease. Therefore, if we do too many iterations we may not get the right answer. This problem can be handled in two ways. The *first one* is running the algorithm above several times with a different number of steps. The *second one* is invoking a different algorithm called «*quantum counting*» to estimate the number of $x_i = 1$ and then choose the number of steps for the search algorithm based on that. Either of those approaches gives us solution to:

Problem	$x_1 \in \{0,1\}, x_2 \in \{0,1\}, \ldots, x_N \in \{0,1\}$
Output	i with $x_i = 1$ if there is one «none» if $x_i = 0$ for all i
Theorem (Boyer, 1998)	There is an algorithm that solves the problem with $O\left(\sqrt{N}\right)$

Remark. Many problems can be solved by reductions to both *Problems* mentioned above. For example, consider the satisfiability that is the canonical NP-compete problem. We have a Boolean formula $F(a_1,\ldots,a_n)$ and we have to find whether there exists a satisfying assignment $\left(a_1 \in \{0,1\}, a_2 \in \{0,1\}, \ldots, a_N \in \{0,1\}\right)$ for which $F(a_1,\ldots,a_n) = 1$. We can reduce the satisfiability to above-mentioned *Problem* by setting $N = 2^n$ and defining $(x_1,\ldots,x_N)$ to be $F(a_1,\ldots,a_n)$ for $N = 2^n$ possible assignments $\left(a_1 \in \{0,1\}, a_2 \in \{0,1\}, \ldots, a_N \in \{0,1\}\right)$. This means that we construct an algorithm that takes $a_1,\ldots,a_n$ and checks if $F(a_1,\ldots,a_n) = 1$. Then if we replace the black-box in the Grover's QSA by this algorithm we get an algorithm that finds a satisfying assignment in the time of $O\left(\sqrt{2^n}\right)$ times needed to check one assignment. A similar reduction applies to any other problem that can be solved by checking all possibilities in some search space.

Computational steps and physical interpretation of Grover's QSA

Let us suppose that we have an unstructured DB with N elements. Without loss of generality suppose that the elements are numbers from 0 to $N-1$. The elements are not ordered. Classically, we would test each element at a time until we hit the one searched for. This takes an average of $\frac{N}{2}$ attempts and N in the worst case, therefore the complexity is $O(N)$. As we will see, using quantum mechanics only $O\left(\sqrt{N}\right)$ trials are needed. For simplicity let's assume that $N=2^n$ for some integer n. Grover's QSA has *two registers*: n q-bits in the first one and one q-bit in the second one.

The *first* step is to create a superposition of all 2^n computational basis states $\left\{|0\rangle,\ldots,|2^n-1\rangle\right\}$ of the first register. This is to be achieved in the following way. Initialize the first register in the state $|00\ldots0\rangle$ and apply the operator $H^{\otimes n}$:

$\|\psi\rangle$	=	$H^{\otimes n}\|00\ldots0\rangle$
	=	$\left(H\|0\rangle\right)^{\otimes n}$
	=	$\left(\frac{\|0\rangle+\|1\rangle}{\sqrt{2}}\right)^{\otimes n}$
	=	$\frac{1}{\sqrt{N}}\sum_{i=0}^{N-1}\|i\rangle$

$|\psi\rangle$ is a superposition of all basis states with equal amplitudes of probability given by $\frac{1}{\sqrt{N}}$.

The *second* register can begin with a state $|1\rangle$ and after a Hadamard gate applied it will be in state $|-\rangle=\frac{1}{\sqrt{2}}\left(|0\rangle-|1\rangle\right)$. Now define $f:\{0,\ldots,N-1\}\to\{,1\}$ as a function that recognizes the solution:

$$f(i)=\begin{cases}1, \textit{if } i \textit{ is the searched element } \left(i_0\right)\\ 0, \textit{otherwise}\end{cases}.$$

This function is used in the classical algorithm. In the QA let us assume that it is possible to build a linear unitary operator also dependent on f, U_f, such that $U_f\left(|i\rangle|j\rangle\right)=|i\rangle|j\oplus f(i)\rangle$. Operator U_f is called *a quantum oracle* and its physical meaning is described below. In the above equation $|i\rangle$ stands for a state of the first register, so i is in the set $\left\{0,\ldots,2^n-1\right\}$, $|j\rangle$ is a state of the second register, so j is in $\{0,1\}$ and the sum is modulo 2. It is easy to check that

$U_f\left(\|i\rangle\|-\rangle\right)$	=	$\frac{1}{\sqrt{2}}\left[U_f\left(\|i\rangle\|0\rangle\right)-U_f\left(\|i\rangle\|1\rangle\right)\right]$
	=	$\frac{1}{\sqrt{2}}\left[\|i\rangle\|f(i)\rangle-\|i\rangle\|1\oplus f(i)\rangle\right]$
	=	$(-1)^{f(i)}\|i\rangle\|-\rangle$

In the last equation we used the fact that

$$1 \oplus f(i) = \begin{cases} 0, & for\ i = i_0 \\ 1, & for\ i \neq i_0 \end{cases}.$$

Now look at what happens when we apply oracle operator U_f to the superposition state coming from the first step $|\psi\rangle|-\rangle$. The state of the second register does not change. Let us call $|\psi_1\rangle$ the resulting state of the first register:

$\|\psi_1\rangle\|-\rangle$	=	$U_f \|\psi\rangle\|-\rangle$
	=	$\frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} U_f \left(\|i\rangle\|-\rangle \right)$
	=	$\frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} (-1)^{f(i)} \|i\rangle\|-\rangle$

$|\psi_1\rangle$ is a superposition of all basis elements but the probability amplitude of the searching element is negative while all others are positive.

Remark. The searched element has been marked with a minus sign. This result is obtained using a feature called *quantum parallelism*. At the quantum level it is possible «to see» all DB elements simultaneously. The position of the searched element is known: it is the value of i of the term with negative amplitude in last equation. This quantum information is not fully available at the classical level. The classical information of a quantum state is obtained by practical measurements and at this point it does not help if we measure the state of the first register because it is much more likely that we obtain a non-desired element instead of the searched one. Before we can perform a measure the next step should be to increase the amplitude of the searched element while decreasing the amplitude of the others. This is quite general: QA's work by increasing the amplitude of the states that carry the desired result. After that a measurement will hit the solution with high probability.

Many QA's can be analyzed in a query (oracle) model where input is given by a block-box (that answers queries) and the complexity of the algorithm is measured by the number of queries to the black-box that it uses.

Example: *Query model.* The majority of QA's have operated in the so-called black-box setting (or DB–query model). In the black-box model the input of the function f what we want to compute can only be accessed by means of queries to a black-box. This returns the $i-th$ bit of the input when queried on i. In the query model the input $x_1,\ldots,x_N$ is contained in a black-box and can be accessed by queries to the black-box. In each query we give i to the black-box and the black-box outputs x_i. The goal is to solve the problem with the minimum number of queries. The classical version of this model is known as *decision trees*. There are two ways to define the query box in the quantum model.

The *first* is an extension of the classical query.

Fig. 3 shows quantum black-box for this case.

It has two inputs i consisting of $\lceil \log N \rceil$ bits and b consisting of 1 bit. If the input to the query box is a basis state $|i\rangle|b\rangle$, the output is $|i\rangle|b \oplus x_i\rangle$. If the input is a superposition $\sum_{i,b} a_{i,b} |i\rangle|b\rangle$ the output is $\sum_{i,b} a_{i,b} |i\rangle|b \oplus x_i\rangle$. Notice that this definition is applied both to the case when the values of x_i are binary and to the case when they are $k-$valued. In the $k-$valued case we just make b consisting of $\lceil \log_2 k \rceil$ bits and take $b \oplus x_i$ to be bit-wise XOR of b and x_i.

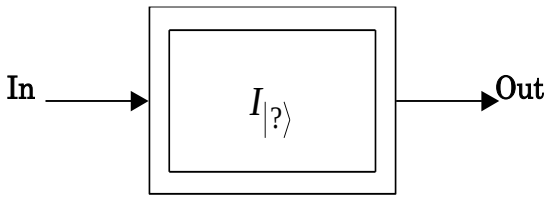


Figure 3. A black-box computing device

The *second* form of quantum query (which only applies to problem with $\{0,1\}-$valued x_i) the black-box has just one input i. If the input is a state $\sum_i a_i|i\rangle$ the output is $\sum_i a_i(-1)^{x_i}|i\rangle$. While this form is less intuitive it is very convenient to use it in QA's including Grover's QSA.

Remark. For this case we consider the first form ~~as~~ to be our main definition but we use the second one when describing Grover's QSA. This is possible to do because ~~a~~ the query of the second type can be simulated by ~~a~~ the query of the first type. Conversely, the oracle of the first type can be simulated by a generalization of the sign oracle that receives $\sum_i a_{i,b}|i\rangle|b\rangle$ as an input and outputs: $\sum_i a_i(-1)^{b\cdot x_i}|i\rangle|b\rangle$. A quantum query model algorithm with T queries is just a sequence of unitary transforms $U_0 \to O \to U_1 \to O \to \ldots U_{T-1} \to O \to U_T$ on some finite-dimensional space $\mathbb{C}^k$, $U_0, U_1, \ldots U_{T-1}, U_T$ can be any unitary transformations that do not depend on the bits $x_1, \ldots, x_N$ inside the black-box. O-s are query transformations that consist of applying the black-box to the first $\log N + 1$ bits of the state. It signifies that we represent basis states of $\mathbb{C}^k$ as $|i,b,z\rangle$. Then O maps $|i,b,z\rangle$ to $|i,b\oplus x_i,z\rangle$. We use O_x to denote the query transformation corresponding to an input $x=(x_1,\ldots,x_N)$.

The computation starts with the state $|0\rangle$. Then we apply $U_0, O_x, \ldots, O_x, U_T$ and measure the final state. The result of the computation is the right most bit of the state obtained by the measurement (or several bits if we are considering a problem where the answer has more than 2 values). The QA computes a function $f(x_1,\ldots,x_N)$ if for every $x=(x_1,\ldots,x_N)$ for which f is defined the probability that the rightmost bit of $U_T O_x U_{T-1} \ldots O_x U_0|0\rangle$ equals $f(x_1,\ldots,x_N)$ is at least $1-\varepsilon$ for some fixed $\varepsilon < \frac{1}{2}$. The query complexity of f is the smallest number of queries used by a QA that computes f. We denote it by $Q(f)$.

Let us consider now the quantum oracle models that in quantum computation are used more in detail.

Quantum oracle model

The Grover's QSA solves the unstructured search problem, under the assumption that there exists a *computational oracle* that can decide whether a candidate solution is the true solution.

Types and relations between oracle models. The following oracles are defined in Table 1 for a general function $f:\{0,1\}^m \to \{0,1\}^n$.

Here x and b are strings of m and n bits respectively, $|x\rangle$ and $|b\rangle$ the corresponding computational basis states and $\oplus$ is addition modulo 2^n. The oracles P_f and S_f are equivalent in power: each of the oracle can be constructed by a quantum circuit containing just one copy of the other.

If we take $m=n$ and suppose we know f is a permutation on the set $\{0,1\}^n$ then M_f is a simple invertible quantum map associated to f.

Table 1. Oracles functions

Number	Title of oracle	Type	Definition

1	The phase oracle	P_f	$\|x\rangle\|b\rangle \to \exp\left\{\frac{2\pi i f(x)\cdot b}{2^n}\right\}\|x\rangle\|b\rangle$
2	The standard oracle	S_f	$\|x\rangle\|b\rangle \to \|x\rangle\|b \oplus f(x)\rangle$
3	The minimal oracle	M_f	$\|x\rangle \to \|f(x)\rangle$

Example: *Each oracle is simulating the other.* One way round turns out to be simple. We can construct S_f from M_f and $(M_f)^{-1} = M_{f^{-1}}$ as follows: $\boxed{S_f = (M_{f^{-1}} \otimes I) \circ A \circ (M_f \otimes I)}$ where «$\circ$» represents the decomposition of operations (or concatenation of networks) and the modulo *N* adder *A* is defined by *A*: $|a\rangle \otimes |b\rangle \to |a\rangle \otimes |a \oplus b\rangle$. Thus, a standard oracle can be simulated given a minimal oracle using just two invocations, one of M_f and one of $(M_f)^{-1}$. However, the converse is not true: simulating a minimal oracle M_f requires exponentially many uses of the standard oracle S_f. First, consider the standard oracle $S_{f^{-1}}$ which maps bits state $|y\rangle|b\rangle$ to $|y\rangle|b \oplus f^{-1}(y)\rangle$. Since $S_{f^{-1}}: |y\rangle|0\rangle \to |y\rangle|f^{-1}(y)\rangle$ stimulation of it allows us to solve the search problem of identifying $|f^{-1}(y)\rangle$ from a DB of *N* elements. It is known that using Grover's search algorithm one can simulate $S_{f^{-1}}$ with $O(\sqrt{N})$ invocations of S_f.

Example. In the following example we explain one possible way of doing that. Prepare the state $|y\rangle|0\rangle|0\rangle|0\rangle$ where first three registers consist of *n* q-bits and the last register is a single q-bit. Apply Hadamard transformations on the second register to get $|\Phi_1\rangle = |y\rangle \sum_{x \in Z_n} |x\rangle|0\rangle|0\rangle$. Invoking S_f on the second and third registers new one gives $|y\rangle\left[\sum_{x \in Z_N} |x\rangle|f(x)\rangle\right]|0\rangle$. Using CNOT gates compare the first and third registers and put the result in the fourth obtaining $\left[|y\rangle \sum_{x \in Z_N, x \neq f^{-1}(y)} |x\rangle|f(x)\rangle|0\rangle\right] + \left[|y\rangle|f^{-1}(y)\rangle|y\rangle|1\rangle\right]$.

Now apply $(S_f)^{-1}$ to the second and third registers obtaining

$$\left(|y\rangle \sum_{x \in Z_N, x \neq f^{-1}(y)} |x\rangle|0\rangle|0\rangle\right) + \left(|y\rangle|f^{-1}(y)\rangle|0\rangle|1\rangle\right).$$

Taken together these operations leave the first and third registers unchanged, while their action on the second and fourth ones defines an oracle for the search problem. Applying Grover's algorithm to this oracle we obtain the state $|y\rangle|f^{-1}(y)\rangle$ after $O(\sqrt{N})$ invocations.

Example: *The oracle model.* Suppose we are supplied with a model *oracle* – a black-box internal workings of which we discuss later but that are not important at this stage – with the ability to recognize solutions to the search problem. This recognition is signaled by making use of an *oracle q-bit*. More precisely the oracle is a unitary operator O defined by its action on the computational basis: $|x\rangle|q\rangle \xrightarrow{O} |x\rangle|q \oplus f(x)\rangle$ where $|x\rangle$ is the index register, $\oplus$ denotes addition modulo 2 and the oracle q-bit $|q\rangle$ is a single q-bit that is flipped if $f(x)=1$ and is being unchanged otherwise. We can check whether x is a solution to our search problem by preparing $|x\rangle|0\rangle$ applying the oracle and checking to see if the q-bit has been flipped to $|1\rangle$. In the QSA it is useful to apply q-bit initially in the state $\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$ just as it was done in the Deutsch –

Jozsa algorithm. If x is not a solution to the search problem applying the oracle to the state $|x\rangle\frac{1}{\sqrt{2}}(|0\rangle-|1\rangle)$ does not change the state.

On the other hand, if x is a solution to the search problem then $|0\rangle$ and $|1\rangle$ are interchanged by the action of the oracle giving a final state $\left[-|x\rangle\frac{1}{\sqrt{2}}(|0\rangle-|1\rangle)\right]$. The action of the oracle is thus:

$$|x\rangle\underbrace{\left(\frac{1}{\sqrt{2}}(|0\rangle-|1\rangle)\right)}_{Oracle\ qubit}\xrightarrow{O}(-1)^{f(x)}|x\rangle\underbrace{\left(\frac{1}{\sqrt{2}}(|0\rangle-|1\rangle)\right)}_{Oracle\ qubit}.$$

Remark. Notice that the state of the oracle q-bit is not changed. It turns out that this remains $\frac{1}{\sqrt{2}}(|0\rangle-|1\rangle)$ throughout the QSA and therefore, it can be omitted from further discussion of the algorithm simplifying the description. With this convention the action of the oracle may be written: $|x\rangle\xrightarrow{O}(-1)^{f(x)}|x\rangle$.

We say that the oracle *marks* the solutions to the search problem by shifting the phase of the solution. For any N item search problem with M solutions it turns out that we need only to apply the search oracle $O\left(\sqrt{\frac{N}{M}}\right)$ times in order to obtain a solution on a quantum computer.

Remark. It seems as though the oracle already knows the answer to the search problem. Question is what possible use could it be to have a QSA based upon such oracle consultants? The answer is that there is a distinction between knowing the solution to a search problem and being able to recognize the solution; the crucial point is that it is possible to do the latter without necessarily being able to do the former.

When we say that one item in search space is marked it's means is given a «*black- box*» or «*oracle*» which has the ability to identify a solution to the search problem when it sees a solution. To say it more precisely, we have *two* registers in our possession. The first register stores the index x to an element in the search space while the second register is a single state z. If supposing s is the marked item then the oracle has the effect: $|x\rangle|z\rangle\to|x\rangle|z\oplus\delta_{sx}\rangle$.

Thus, the oracle «recognized» solutions to the search problem in the sense that it flips the second register when it finds the solution to the problem in the first register. It means that the oracle does *not know* the identity of the state it is searching for but rather can recognize the solution when sees it.

Before describing the steps of the algorithm it's actually very useful to notice two things. First of all, imagine that we prepare the first register in the state $|x\rangle$ and the second register in the superposition $|0\rangle-|1\rangle$. Then the effect of the oracle will be as follows:

$$|x\rangle(|0\rangle-|1\rangle)\to(-1)^{\delta_{sx}}|x\rangle(|0\rangle-|1\rangle).$$

Notice that the state of the second register is left alone by this operation; henceforth we will ignore the state of the second register and will just write the action of the oracle as $|x\rangle\to(-1)^{\delta_{sx}}|x\rangle$.

In a similar way it's useful for us to be able to perform an operation that leaves the state of our register $|x\rangle$ alone unless it is in all zero state in that case a phase shift of (-1) is applied. The computational complexity of the function f is measured by the required number of queries. In this setting we want QA that use significantly fewer queries than the best classical algorithms.

Our purpose is to find the «target» y with the smallest possible number of the oracle evaluations called the *query complexity*. It is remarkable that there is a QSA that enables this search method to be speed-up substantial requiring only $O\left(\sqrt{N}\right)$ operations.

Remark. Elementary probability theory shows that classically if we examine *k* records then we have probability *k/N* of finding the special one, so we need *O* (*N*) such trials to find it with any constant (independent of *N*) level of probability. Grover's quantum algorithm achieves this result with only $O(\sqrt{N})$ steps (or more precisely $O(\sqrt{N})$ iterations of Grover's operator G but $O(\sqrt{N}\log N)$ steps, the $\log N$ term coming from the implementation of H). It may be shown (Zalka, 1997) that the square root speedup of Grover's algorithm is *optimal* within the context of quantum computation.

In Grover's QSA the N inputs are mapped onto the states of n q-bits. The Grover's QSA is optimal exactly and not only asymptotically, optimal for query complexity if quantum computation consists only of unitary transformations with fixed structures and the final measurement.

Thus, the quantum problem becomes one of maximizing the overlap between the state of these n q-bits and the target state $|y\rangle$. This is equivalent to maximizing the probability of obtaining the desired state upon measurement. The initial state of these q-bits is taken to be an equal superposition of all possible bit stings. The Grover operator that is used repeatedly in the algorithm corresponds to a small rotation in the two-dimensional subspace spanned by the initial and target states. Each such rotation requires a single evaluation of $f(x)$. Thus, unlike a classical search the quantum search monotonically rotates the state towards the target.

The circuit for Grover's QSA

Now we shall work out the details by introducing the circuit for Grover's QSA and analyzing it step by step.

Fig. 4 shows the circuit for Grover's QSA.

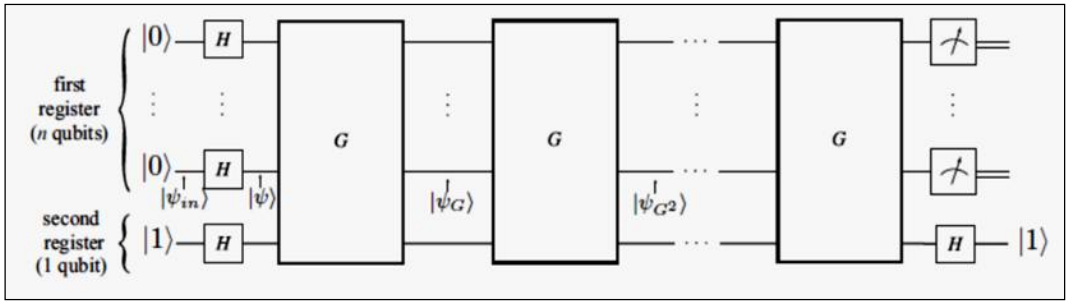


Figure 4. Outline of Grover's algorithm

The unitary operator G is applied $O\left(\sqrt{N}\right)$ times. The exact number will be obtained later on. The circuit for one Grover iteration G is given in Fig. 5.

The states $|\psi\rangle$ and $|\psi_1\rangle$ are given above. The operator $2|\psi\rangle\langle\psi|-I$ is called *inversion about the mean* for reasons that will be clear below. We will also show how each Grover operator raises the amplitude of the searching element: $|\psi_1\rangle$ can be rewritten as $|\psi_1\rangle=|\psi\rangle-\frac{2}{\sqrt{2^n}}|i_0\rangle$, where $|i_0\rangle$ is the searching element. $|i_0\rangle$ is a state of computational basis. Note that $\langle\psi|\,i_0\rangle=\frac{1}{\sqrt{2^n}}$.

Let us calculate $|\psi_G\rangle$ in Fig. 5. Using the abovementioned approach and two last expressions for $|\psi_1\rangle$ and $\langle\psi|\,i_0\rangle$, we obtain

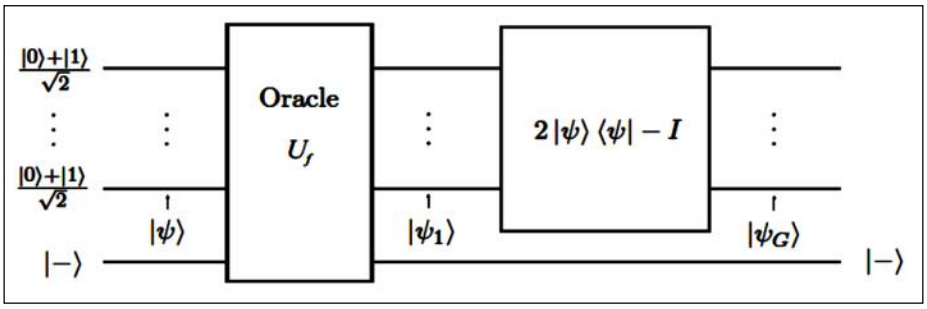


Figure 5. One Grover iteration (G)

The states of the first register correspond to the first iteration

$\lvert\psi_G\rangle$	=	$\left(2\lvert\psi\rangle\langle\psi\rvert - I\right)\lvert\psi_1\rangle$
	=	$\frac{2^{n-2}-1}{2^{n-2}}\lvert\psi\rangle + \frac{2}{\sqrt{2^n}}\lvert i_0\rangle$

This is the state of first register after one application of G; the second register is in the state $|-\rangle$. This allows a nice geometrical representation taking $|i_0\rangle$ and $|\psi\rangle$ as base vectors (non-orthogonal basis). Fig. 6 shows the vectors $|i_0\rangle$ and $|\psi\rangle$.

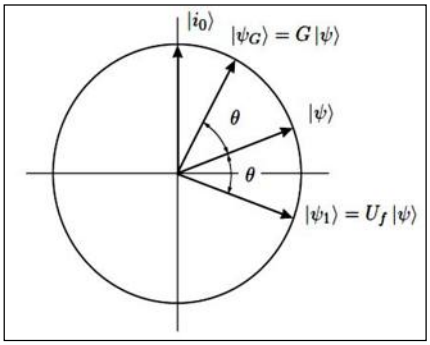


Figure 6. The state of the first register lives in the real vector space spanned by $|i_0\rangle$ and $|\psi\rangle$

We take these states as a basis to describe what happens in Grover's algorithm. They form an angle smaller than 90^0 as can be seen from the relation $\langle\psi|i_0\rangle = \frac{1}{\sqrt{2^n}}$, since $0 < \langle\psi|i_0\rangle < 1$. If n is large, then the angle is nearly 90^0. We can think that $|\psi\rangle$ is the initial state of the first register, and the steps of the computation are the applications of the unitary operators U_f and $2|\psi\rangle\langle\psi| - I$. Then $|\psi\rangle$ will rotate in the real plane spanned by $|\psi\rangle$ and $|i_0\rangle$, keeping the unit norm. This means that the tip of $|\psi\rangle$'s vector lies in the unit circle. From the expressions for $|\psi_1\rangle$ and $\langle\psi|i_0\rangle$ we see that $|\psi\rangle$ rotates θ degrees clockwise, where $\cos\theta = 1 - \frac{1}{2^{n-1}}$.

Fig. 6 shows the position of vector $|\psi_1\rangle$ in the unit circle. From the expressions similar to $\langle\psi|i_0\rangle$ we see that the angle between $|\psi_G\rangle$ and $|\psi\rangle$ is $\cos\theta' = \langle\psi|\psi_G\rangle = 1 - \frac{1}{2^{n-1}}$. So, $\theta' = \theta$ and $|\psi_1\rangle$ rotates 2θ degrees counterclockwise (in the direction of $|i_0\rangle$).

Fig. 7 explains also the placement of $|\psi_G\rangle$.

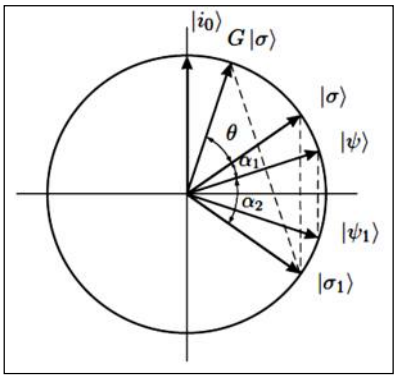


Figure 7. A generic vector $|i_0\rangle$ is reflected around the horizontal axis by the application of U_f, yielding $|\sigma_1\rangle$. Then, the reflection of $|\sigma_1\rangle$ about the mean $|\psi\rangle$ gives $G|\sigma\rangle$, which is θ degrees closer to $|i_0\rangle$ (vertical axis)

Remark. This is remarkable result, since the resulting action of $G=(2|\psi\rangle\langle\psi|-I)U_f$ rotates $|\psi\rangle$ towards $|i_0\rangle$ by θ degrees. This means that the amplitudes of $|i_0\rangle$ in $|\psi_G\rangle$ increased and the amplitudes of $|i\rangle$, $i\neq i_0$, decreased with respect to their original values in $|\psi\rangle$. A measurement, at this point, will return $|i_0\rangle$ more likely than before. But it is not enough in general, since θ is a small angle if $n\gg 1$ while $\cos\theta=1-\frac{1}{2^{n-1}}$. That is why we need to apply G repeatedly, ending up θ degrees closer to $|i_0\rangle$ each time, until the state of the first register be very close to $|i_0\rangle$, so we can measure.

Example: Computation in Grover's quantum gate and geometrical interpretation of simulation results for $N=8$. We will describe Grover's QSA for search space of 8 elements for an unknown record with the unknown label $x_0 = 5$. If $N=8$ then number of input qubit is $n=3, 2^3=8$. There are 3 qubits in the first register and 1 qubit in the second register. For $N=8$, the operator G will be applied two times as we will see from estimation $\left\lceil \frac{\pi}{4}\sqrt{N} \right\rceil$. Fig. 8 shows the circuit in this case.

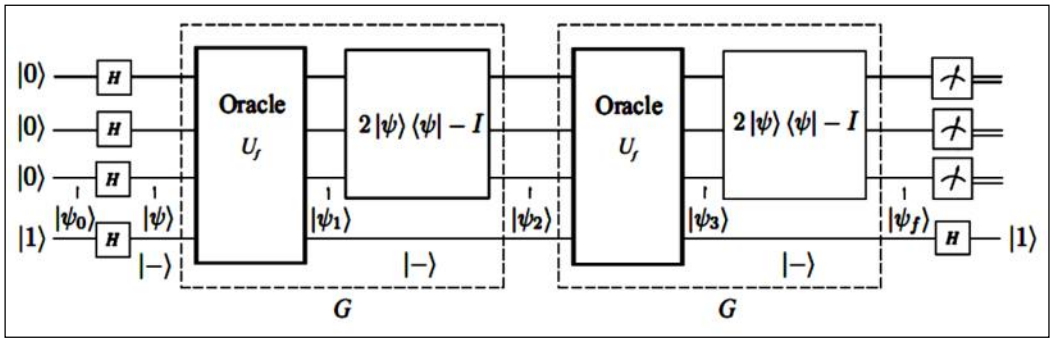


Figure 8. Grover's algorithm for N = 8

Classically, an average of more than 4 queries are needed in order to have a probability of success of more than $\frac{1}{2}$.

1. We are given a black-box computing device (see Fig. 3) that implements as an oracle the unknown unitary transformation

$$U_f = I_{|x_0\rangle} = I_{|5\rangle} = \left(\begin{array}{cccc:cccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ \hdashline 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right).$$

Remark. We cannot open the black-box in Fig. 3 to see what is inside. So we do not know what $I_{|x_0\rangle}$ and x_0 are. The only way that we can glean some information about x_0 is to apply some chosen state $|\psi\rangle$ as input, and then make use of the resulting output. Using of the black-box in Fig. 3 as a component device, we construct a computing quantum gate, which implements the unitary operator

$$Q = -HI_{|0\rangle}HI_{|x_0\rangle} = \frac{1}{4}\left(\begin{array}{cccc:cccc} -3 & 1 & 1 & 1 & -1 & 1 & 1 & 1 \\ 1 & -3 & 1 & 1 & -1 & 1 & 1 & 1 \\ 1 & 1 & -3 & 1 & -1 & 1 & 1 & 1 \\ 1 & 1 & 1 & -3 & -1 & 1 & 1 & 1 \\ \hdashline 1 & 1 & 1 & 1 & 3 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -3 & 1 & 1 \\ 1 & 1 & 1 & 1 & -1 & 1 & -3 & 1 \\ 1 & 1 & 1 & 1 & -1 & 1 & 1 & -3 \end{array}\right).$$

We do not know what unitary transformation *Q* is implemented by the quantum gate because the black-box is one of its essential components. We can compute the state $|5\rangle$ in standard Grover's QSA as following.

STEP 0: We begin by preparing the known state (superposition)

$$|\psi_0\rangle = H|0\rangle = \frac{1}{\sqrt{8}}(1,1,1,1,1,1,1,1)^{transpose}$$

STEP 1: We proceed to loop $K = round\left(\frac{\pi}{4\sin^{-1}(1/\sqrt{8})} - \frac{1}{2}\right) - \frac{1}{2}) = 2$ times in *STEP 1*.

Iteration 1. On the first iteration, we obtain the unknown state (entanglement state)

$$|\psi_1\rangle = Q|\psi_0\rangle = \frac{1}{4\sqrt{2}}(1,1,1,1,5,1,1,1)^{transpose}$$

Iteration 2: On the second iteration, we obtain the unknown state (interference mode)

$$|\psi_2\rangle = Q|\psi_1\rangle = \frac{1}{4\sqrt{2}}(-1,-1,-1,-1,\ 11,-1,-1,-1)^{transpose}$$

and branch to *STEP 2*.

STEP 2: We measure the unknown state $|\psi_2\rangle$ to obtain either $|5\rangle$ with probability

$$Prob_{success} = \sin^2\left[(2K+1)\beta\right] = \frac{121}{128} = 0.9453$$

or some other state with probability

$$Prob_{failure} = \cos^2\left[(2K+1)\beta\right] = \frac{7}{128} = 0.0547$$

and then exit.

2. Let us describe the quantum computation process state at each step shown in the circuit in Fig. 1.27 as following:

$$\left(\left|\psi_0\right\rangle \to \left|\psi\right\rangle \to \left|\psi_1\right\rangle \to \left|\psi_2\right\rangle \to \left|\psi_3\right\rangle \text{ and } \left|\psi_f\right\rangle\right).$$

(1). The *initial* state is $\left|\psi_0\right\rangle = \left|000\right\rangle$;

(2). After Hadamard gates, $\left|\psi\right\rangle = H^{\otimes 3}\left|000\right\rangle = \left(H\left|0\right\rangle\right)^{\otimes 3} = \frac{1}{2\sqrt{2}}\sum_{i=0}^{7}\left|i\right\rangle$;

Suppose that we are searching for the element with index 5.

(3). Since $\left|5\right\rangle = \left|101\right\rangle$,

$$\begin{aligned} U_f\left(\left|101\right\rangle\left|-\right\rangle\right) &= -\left|101\right\rangle\left|-\right\rangle, \text{ for } i = 5 \\ U_f\left(\left|i\right\rangle\left|-\right\rangle\right) &= \left|101\right\rangle\left|-\right\rangle, \text{if } \ i \neq 5 \end{aligned}.$$

Define $\left|u\right\rangle$ as

$$\left|u\right\rangle = \frac{1}{\sqrt{7}}\sum_{i=0, i\neq 5}^{7}\left|7\right\rangle = \frac{\left|000\right\rangle + \left|001\right\rangle + \left|010\right\rangle + \left|011\right\rangle + \left|100\right\rangle + \left|110\right\rangle + \left|111\right\rangle}{\sqrt{7}}.$$

Then $\left|\psi\right\rangle = \frac{\sqrt{7}}{2\sqrt{2}}\left|u\right\rangle + \frac{1}{2\sqrt{2}}\left|101\right\rangle$.

With this result we can see the direction of $\left|\psi\right\rangle$.

Fig. 9 shows this direction of $\left|\psi\right\rangle$.

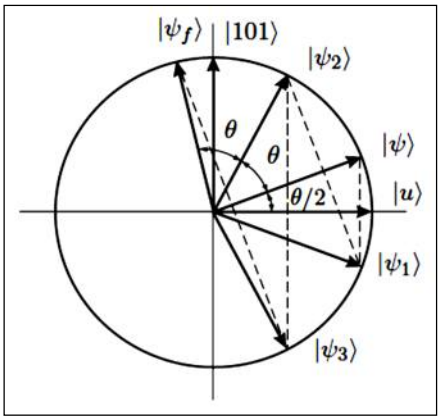


Figure 9. Intermediate states in Grover's algorithm for N = 8

The value of θ is

θ	=	$2\arccos\left(\frac{\sqrt{7}}{2\sqrt{2}}\right)$
	=	$\arccos\left(\frac{3}{4}\right)$
	≈	$41,4^0$

Notice how close is $|\psi_f\rangle$ to $|101\rangle$, indicating a high probability that a measurement will give the searched element. The value of θ is around 41.4º.

(4). The next step is

$$|\psi_1\rangle|-\rangle \;=\; U_f\left(|\psi\rangle|-\rangle\right)= \left(\frac{|000\rangle+|001\rangle+|010\rangle+|011\rangle+|100\rangle-|101\rangle+|110\rangle+|111\rangle}{2\sqrt{2}}\right)|-\rangle$$

Note that $|101\rangle$ is the only with a minus sign. We can write $|\psi_1\rangle$ as

$$|\psi_1\rangle=|\psi\rangle-\frac{1}{\sqrt{2}}|101\rangle \text{ or } |\psi_1\rangle=\frac{\sqrt{7}}{2\sqrt{2}}|u\rangle-\frac{1}{\sqrt{2}}|101\rangle$$

The form of last two equations is useful in the next step of calculation since we have to apply $\left(2|\psi\rangle\langle\psi|-I\right)$. The form in last equation is useful to draw the geometrical state $|\psi_1\rangle$.

Fig. 9 shows the state $|\psi_1\rangle$. $|\psi_1\rangle$ is the reflection of $|\psi\rangle$ with respect to $|u\rangle$.

Next step is the calculation $|\psi_2\rangle=\left(2|\psi\rangle\langle\psi|-I\right)|\psi_1\rangle$. Using the last expressions for $|\psi_1\rangle$, we get $|\psi_2\rangle=\frac{1}{\sqrt{2}}|\psi\rangle+\frac{1}{\sqrt{2}}|101\rangle$ and, using the last expression for $|\psi\rangle$,

$$|\psi_2\rangle=\frac{\sqrt{7}}{4\sqrt{2}}|u\rangle+\frac{5}{4\sqrt{2}}|101\rangle.$$

Let us conform that the angle between $|\psi\rangle$ and $|\psi_2\rangle$ is θ:

$$\cos\theta=\langle\psi|\psi_2\rangle=\frac{1}{2}\langle\psi|\psi\rangle+\frac{1}{\sqrt{2}}\langle\psi|101\rangle=\frac{3}{4},$$

which agrees with the above expression of θ. This completes one application of G.

(5). The second and last application of G is similar. $|\psi_3\rangle$ is given by

$$|\psi_3\rangle=\frac{\sqrt{7}}{2\sqrt{2}}|u\rangle-\frac{5}{4\sqrt{2}}|101\rangle.$$

Using $|\psi\rangle=\frac{\sqrt{7}}{2\sqrt{2}}|u\rangle+\frac{1}{2\sqrt{2}}|101\rangle$, we have

$$|\psi_3\rangle=\frac{1}{2}|\psi\rangle-\frac{3}{2\sqrt{2}}|101\rangle.$$

$|\psi_3\rangle$ is the reflection of $|\psi_2\rangle$ with respect to $|u\rangle$.

(6). The last step is

$$\left|\psi_f\right\rangle = \left(2\left|\psi\right\rangle\left\langle\psi\right| - I\right)\left|\psi_3\right\rangle .$$

Using $\left|\psi\right\rangle = \frac{\sqrt{7}}{2\sqrt{2}}\left|u\right\rangle + \frac{1}{2\sqrt{2}}\left|101\right\rangle$ and $\left|\psi_3\right\rangle = \frac{1}{2}\left|\psi\right\rangle - \frac{3}{2\sqrt{2}}\left|101\right\rangle$, we have

$$\left|\psi_f\right\rangle = \frac{\sqrt{7}}{8\sqrt{2}}\left|u\right\rangle + \frac{11}{8\sqrt{2}}\left|101\right\rangle .$$

It is easy to conform that $\left|\psi_f\right\rangle$ and $\left|\psi_2\right\rangle$ form an angle θ. Note that the amplitude of the state $\left|101\right\rangle$ is much bigger than the amplitude of any other state $\left|i\right\rangle$ $\left(i \neq 5\right)$ in last expression for $\left|\psi_f\right\rangle$. This is the way most QA work. They increase the amplitude of the states that carry the desired information. A measurement of the state $\left|\psi_f\right\rangle$ in the computational basis will project it into the state $\left|101\right\rangle$ in the computational basis with probability $p = \left|\frac{11}{8\sqrt{2}}\right|^2 \approx 0.9453$. The chance of getting the result $\left|101\right\rangle$, which reads as number 5, is around $94{,}5\%$.

Example: *Generalization of computational process in QSA.* The easiest way to calculate the output of Grover's QSA is to consider only the action of G instead of breaking the calculation into action of the oracle U_f and the inversion about the mean. To this end, we choose $\left|i_0\right\rangle$ and $\left|u\right\rangle$ as the basis for the subspace where $\left|\psi\right\rangle$ rotates after successive applications of G. $\left|i_0\right\rangle$ is the searched state and $\left|u\right\rangle$ is defined from the above expression in general form as

$$\left|u\right\rangle = \frac{1}{\sqrt{N-1}}\sum_{i=0, i\neq i_0}^{N-1}\left|i\right\rangle = \sqrt{\frac{N}{N-1}}\left|\psi\right\rangle - \frac{1}{\sqrt{N-1}}\left|i_0\right\rangle .$$

From the first expression above we easily see that $\left\langle i_0\right|\left.u\right\rangle = 0$, i.e., $\left|i_0\right\rangle$ and $\left|u\right\rangle$ are orthogonal. From the second equation we have $\left|\psi\right\rangle = \sqrt{1-\frac{1}{N}}\left|u\right\rangle + \frac{1}{\sqrt{N}}\left|i_0\right\rangle$. The state of the quantum computing at each step is $G^k\left|\psi\right\rangle = \cos\left(\frac{2k+1}{2}\theta\right)\left|u\right\rangle + \sin\left(\frac{2k+1}{2}\theta\right)\left|i_0\right\rangle$, where we have dropped the state of the second register it is $\left|-\right\rangle$ all the time.

Fig. 10 shows effect of G on $\left|\psi\right\rangle$.

The above last equation is obtained after analyzing the components of $G^k\left|\psi\right\rangle$. The value of θ is obtained substituting k for 0 in last expression and comparing it above with two last equations, $\theta = 2\arccos\sqrt{1-\frac{1}{N}}$. The equation for $G^k\left|\psi\right\rangle$ expresses the fact (we proved above), that each application of G rotates the state of the first register by θ degrees towards $\left|i_0\right\rangle$. Figure 1.29 shows successive applications of G.

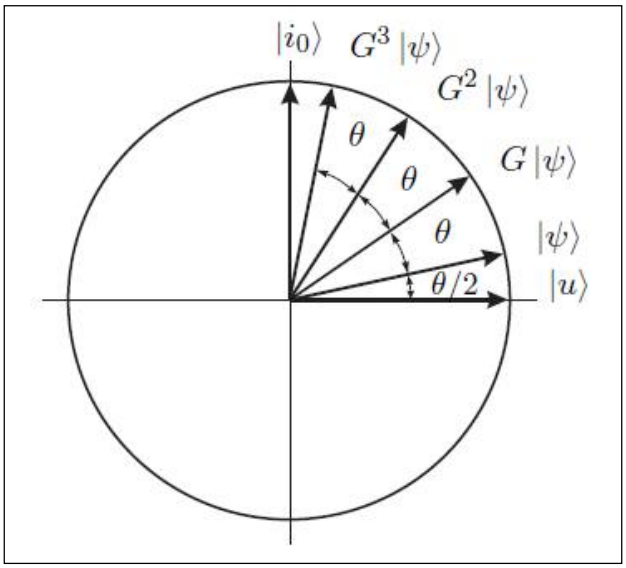


Figure 10. Effect of G on $|\psi\rangle$

The number of times k_0 that G must be applied obeys the equation $k_0\theta+\frac{\theta}{2}=\frac{\pi}{2}$. Since k_0 must be integer, we write $k_0 = round\left(\frac{\pi-\theta}{2\theta}\right)$, where θ is define from above equation $\theta = 2\arccos\sqrt{1-\frac{1}{N}}$. If $N \gg 1$, by Teilor expanding this last equation, we get $\theta \approx \frac{2}{\sqrt{N}}$ and from the expression for $k_0 = round\left(\frac{\pi-\theta}{2\theta}\right)$, and we have $k_0 = round\left(\frac{\pi}{4}\sqrt{N}\right)$. After applying k_0 times the operator G, the probability p of finding the desired element (after a measurements) is

$$p = \sin^2\left(\frac{2k_0+1}{2}\theta\right).$$

Example: *Probability of successful result of quantum search.* Fig. 11 shows the evolution value of probability p of finding the desired element (after a measurements) for n form 2 to 30.

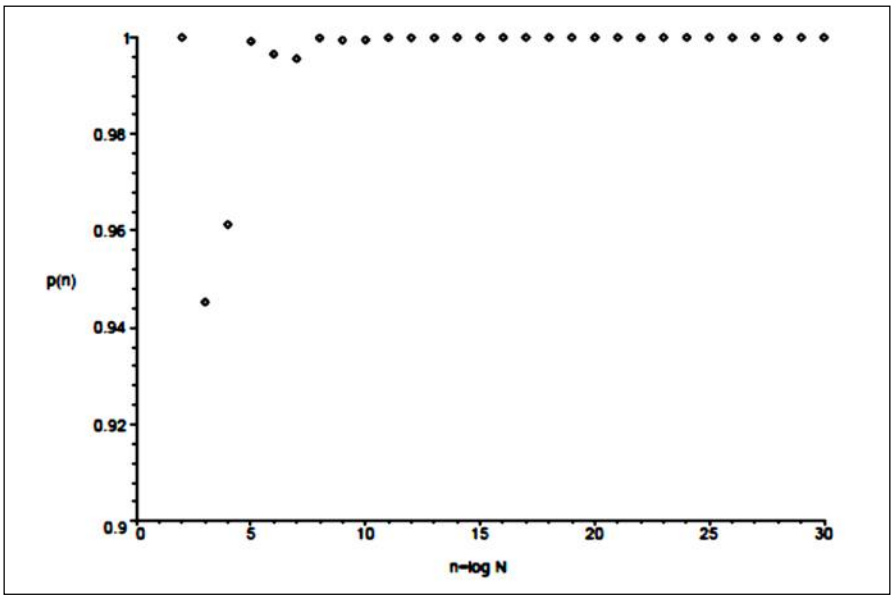


Figure 11. Probability of succeeding as a function of n

Recall that $N = 2^n$, so for $n = 30$ the search space has around one billion elements. For $n = 2$ the probability of getting the result is exactly 1. The reason for this case is that the equation for θ is $\theta = 2\arccos\sqrt{1-\frac{1}{N}}$ and yields $\theta = \frac{\pi}{3}$. And $|\psi\rangle$ makes an angle $\theta = \frac{\pi}{6}$ with $|u\rangle$.

Applying G one times rotates $|\psi\rangle$ to $|i_0\rangle$ exactly. For $n = 2$, from $p = \sin^2\left(\frac{2k_0+1}{2}\theta\right)$ yields $p \approx 0,9453$ which is the result that described above.

Remark. The description of Shor's quantum search algorithm is closed to the solution of quantum cryptography problems and in part 2 of this book is considered.

Shor's quantum algorithm for factoring

Shor's algorithm is the most important algorithmic result in quantum computing. The algorithm is built on ideas that already appear in Deutsch and Jozsa's algorithm and in Simon's algorithm, and like these algorithms the basic ingredient of the algorithm is the Fourier Transform (FT) will be stated as follows:

Input:	*An integer N*
Output:	*A non-trivial factor of N, if it exists*

Remark. There is no proof that there is no polynomial classical factorization algorithm. The problem is even not known to be NP-complete. However, factorization is regarded so hard because many works have tried to solve it efficiently and failed. In 1994 Shor published a polynomial (in log(*N*)) QA for solving this problem. In fact, Shor presented a QA not for factoring but for a different problem:

Order modulo *N*:	
Input:	*An integer N and Y are coprimes to N*
Output:	*The order of Y, i.e. the minimal positive integer r such that* $Y^r = 1 \bmod N$.

In order to factor a number *N* it is enough to be able to find the order of *x* in Z_N.

Let us consider theoretical aspects of Shor' algorithm.

Reduction of factorization to order-finding

Let us describe Shor's algorithm for finding the prime factors of a composite number *N*. Think of a large number such as one with 300 digits in decimal notation, since such numbers are used in cryptography. Though *N* is large the number of q-bits necessary to store is small. In general $\log_2 N$ is not an integer, so let us define $\lceil \log_2 N \rceil$. A quantum computer with *n* q-bits can store *N* or any other positive integer less than *N*. With a little thought we see that the number of prime factors of *N* is at most *n*. If both the number of q-bits and the number of factors are less than or equal to *n*, then it is natural to ask if there is an algorithm that factors *N* in a number of steps which is polynomial in *n*.

More technically, the question is: is there a factorization algorithm in the complexity class P?

Reduction of factorization of *N* to the problem of finding the order of an integer *x* less than *N* is as follows. If *x* and *N* have common factors, then *gcd*(*x*,*N*) gives a factor of *N*, therefore it suffices to investigate the case when *x* is coprime to *N*. The order of *x* mod *N* is defined as the least positive integer *r* such that $x^r \equiv 1 \bmod N$. If *r* is even, we can define *y* by $x^{r/2} \equiv y \bmod N$.

Remark. The above notation means that *y* is the remainder of $x^{r/2}$ divided by *N* and by definition $0 \le y < N$. Note that *y* satisfies $y^2 \equiv 1 \bmod N$ or equivalently $(y-1)(y+1) \equiv 0 \bmod N$, what means that *N* divides $(y-1)(y+1)$. If $1 < y < N-1$ the factors $y-1$ and $y+1$ satisfy $0 < y-1 < y+1 < N$, therefore *N* cannot divide *y*

-1 nor $y+1$ separately. The only alternative is that both $y-1$ and $y+1$ have factors of N (that yield N by multiplication) (see in details Appendix: *Elements of Number Theory*).

So, $gcd(y-1, N)$ and $gcd(y+1, N)$ yield non trivial factors of N (*gcd* stands for the greatest common divisor). If N has remaining factors they can be calculated applying the algorithm recursively.

Example. Consider $N = 21$ as an example. The sequence of equivalences

$$2^4 \equiv 16 \bmod 21, \quad 2^5 \equiv 11 \bmod 21, \quad 2^6 \equiv 11 \times 2 \equiv 1 \bmod 21,$$

show that the order of 2 mod 21 is $r = 6$.

Therefore, $y \equiv 2^3 \equiv 8 \bmod 21$, $y - 1$ yields the factor 7 and $y + 1$ yields the factor 3 of 21.

In summary, if we pick up at random a positive integer x less than N and calculate $gcd(x, N)$ then either we have a factor of N or we learn that x is coprime to N. In the latter case if x satisfies the conditions (1) its order r is even and (2) $0 < y - 1 < y + 1 < N$ then $gcd(y - 1, N)$ and $gcd(y + 1, N)$ yield factors of N. If one of the conditions is not true we start over until finding a proper candidate x. The method would not be useful if these assumptions were too restrictive but fortunately that ~~is~~ was not the case. The method systematically fails if N is a power of some odd prime but an alternative efficient classical algorithm for this case is known. If N is even we can keep dividing by 2 until the result turns out to be odd. It remains to apply the method for odd composite integers that are not a power of some prime number. It is cumbersome to prove that the probability of finding x coprime to N satisfying the conditions (1) and (2) is high; in fact this probability is $1-1/2^{k-1}$ where k is the number of prime factors of N. In the worst case (N has 2 factors) the probability is greater than or equal to ½.

At first sight, it seems that we have just described an efficient algorithm to find a factor of N. That is not true, since it is not known an efficient classical algorithm to calculate the order of an integer x mod N. On the other hand, there is (after Shor's work) an efficient QA.

Let us describe it.

Theorem: If there is a polynomial (in log N) algorithm to solve order modulo N then there is a polynomial algorithm to solve factorization.

Proof: We show a (classical probabilistic) reduction from FACTORING to ORDER. That is, we assume we have a black box algorithm for finding the order of a given integer x (coprime to N), in Z_N.

Lemma 1. A solution to the equation $x^2 \equiv 1 (\bmod N)$ with $x \neq -1, +1$ gives a factor of N.
This is true since this equality implies $(x + 1)(x - 1) = mN$ for some integer m, and both $1 < x-1, x+1 < N-1$. This means that either $\gcd(x-1, N) \neq 1, N$ or $\gcd(x-1, N) \neq 1, N$, as N divides $(x + 1)(x - 1)$. This proves the lemma.

Now pick a random (nonzero) element of Z_N. If $gcd(x, N) > 1$, then we have found a factor of N. If not, find the order r of x in Z_N. Assume we are lucky, and r is even and also $y := x^{r/2} \neq -1, +1$. Then y is a nontrivial solution to the equation $y^2 \equiv 1 (\bmod N)$, which gives us a factorization of N by the previous lemma. What is the probability that we are lucky?

Lemma 2. Let $N = p_1^{\alpha_1} \cdot p_2^{\alpha_2} \ldots \cdot p_m^{\alpha_m}$, where p_i are distinct primes, N is odd and not prime (i.e. m > 1, $p_i \neq 2$). Choose $x \in Z_N$ randomly. Then

$$\text{Prob}\ (ORD_{Z_N}(x) \text{ is even, and } x^{r/2} \neq -1, +1) \geq 1 - 2^{-m} \geq 1/2$$

We shall not prove this lemma (proof can be found in Nielsen-Chuang (2000)). This shows that a randomly chosen x will give a factor of x using the procedure described above with probability ≥ ½. This can, as usual, be amplified by repeating this process several times. We have thus proved theorem.

Example. Reduction to Period-Finding. Shor's algorithm finds a factor by finding the period of some sequence. We first show how efficient period-finding suffices for efficient factoring. Suppose we want to find factors of the composite number $N > 1$. Randomly choose some integer $x \in \{2, \ldots, N-1\}$. Consider the sequence

$$1 = x^0 \bmod N, x^1 \bmod N, x^2 \bmod N, ...$$

This sequence will cycle after a while: there is a least $0 < r \le N$ such that $x^r = 1 \bmod N$. This *r* is called the *period* of the sequence. It can be shown that with probability ≥ ¼, *r* is even and $x^{r/2}+1$ and $x^{r/2}-1$ are not multiples of *N*. In that case:

x^r	≡	$1 \bmod N$	⇔
$(x^{r/2})^2$	≡	$1 \bmod N$	⇔
$(x^{r/2}+1)(x^{r/2}-1)$	≡	$0 \bmod N$	⇔
$(x^{r/2}+1)(x^{r/2}-1)$	=	*kN*	*for some k.*

Not that $k > 0$ because both $x^{r/2}+1 > 0$ and $x^{r/2}-1 > 0$ ($x > 1$). Hence $x^{r/2}+1$ or $x^{r/2}-1$ will share a factor with *N*. Because $x^{r/2}+1$ and $x^{r/2}-1$ are not multiples of *N* this factor will be <*N*, and in fact *both* these numbers will share a non-trivial factor with *N*. Accordingly, if we have *r* then we can efficiently (in $\tilde{O}(\log N)$ steps) compute the greatest common divisors $gcd(x^{r/2}+1, N)$ and $gcd(x^{r/2}-1, N)$, and both of these two numbers will be non-trivial factors of *N*. If we are unlucky we might have chosen an *x* that does not give a factor (which we can detect efficiently), but trying a few different random *x* gives a high probability of finding a factor.

Thus the problem of factoring reduces to finding *r*.

We will show below how the QFT enables us to do this.

Shor's algorithm for finding the order

Given *N* choose a random (with the uniform distribution) *m* ($1 < m \le N$). We assume that $gcd(m, N) = 1$, otherwise we would already know a divisor of *N*. We want to find the order of *m*, i.e. the least integer *r* such that

$$m^r \equiv 1 (\bmod N).$$

Fix some *q* of the form $q = 2^s$ with $N^2 \le q < 2N^2$. The algorithm will use the Hilbert space

$$H = C^q \otimes \boxed{C^{N_1}} \otimes C^k$$

where C^q and C^{N_1} are two quantum registers which hold integers represented in binary. Here N_1 is an integer of the form $N_1 = 2^l$ for some *l* such that $N \le N_1$. There is also the work space C^k to make arithmetical operations.

We will not indicate it explicitly. If $a = a_0 2^0 + 2^1 a_1 + 2^2 a_2 + ... + 2^s a_s$ is the binary representation ($a_i = 0,1$) of an integer *a* then we write $|a\rangle = |a_0\rangle \otimes ... \otimes |a_s\rangle$ where $|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$, $|1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$ is the basis in the two dimensional complex space C^2. We have the data (*N*, *m*, *q*).

As abovementioned, the algorithm for finding the order *r* of *m* consists from 5 steps:

1.	Preparation of quantum state.
2.	Modular exponentiation.
3.	Quantum Fourier transform.
4.	Measurement.
5.	Computation of the order at the classical computer.

Description of the algorithm

Step 1: *Preparation of quantum state.* Put the first register in the uniform superposition of states representing numbers $a(\mathrm{mod}\, q)$. The quantum computer will be in the state $|\psi_1\rangle = \frac{1}{\sqrt{q}}|a> \otimes |0>$.

Step 2: *Modular exponentiation.* Compute $m^a (\mathrm{mod}\, N)$ in the second register. This leaves the quantum computer in the state

$$|\psi_2\rangle = \frac{1}{\sqrt{q}} \sum_{a=0}^{q-1} |a> \otimes |m^a (\mathrm{mod}\, N)>.$$

Step 3: *Quantum Fourier transform* (QFT). Perform the QFT on the first register, mapping $|a>$ to $\frac{1}{\sqrt{q}} \sum_{c=0}^{q-1} e^{2\pi i a c/q} |c>$. The quantum computer will be in the state

$$|\psi_3\rangle = \frac{1}{q} \sum_{c=0}^{q-1} \sum_{c=0}^{q-1} e^{2\pi i a c/q} |c\rangle \otimes |m^a (\mathrm{mod}\, N)\rangle.$$

Step 4: *Measurement.* Make the measurement on both registers $|c\rangle$ and $|m^a (\mathrm{mod}\, N)\rangle$.

Remark To find the period *r* we will need only the value of $|c\rangle$ in the first register but for clarity of computations we made the measurement on the both registers. The probability $P(c, m^k (\mathrm{mod}\, N))$ that the quantum computing ends in a particular state

$$|c; m^k (\mathrm{mod}\, N)\rangle = |c\rangle \otimes |m^k (\mathrm{mod}(N)\rangle$$

according to quantum mechanics law is

$$P(c, m^k (\mathrm{mod}\, N)) = |< m^k (\mathrm{mod}\, N); c|\psi_3 >|^2$$

where we can assume $0 \le k < r$.

We will use the following Theorem, which shows that the probability $P(c, m^k (\mathrm{mod}\, N))$ is large if the residue of $rc(\mathrm{mod}\, q)$ is small. Here *r* is the order of *m* in the group $(Z / NZ)^*$ of residues of modulo *N*.

Theorem: If there is an integer *d* such that $-\frac{r}{2} \le rc - dq \le \frac{r}{2}$ and *N* is sufficiently large then

$$P(c, m^k (\mathrm{mod}\, N)) \ge \frac{1}{3r^2}$$

Step 5: *Computation of the order at the classical computer*. We know *N*, *c* and *q* and we want to find the order *r*. Because $q > N^2$, there is at most one fraction *d/r* with $r < N$ that satisfies the inequality. We can obtain the fraction *d/r* in lowest terms by rounding *c/q* to the nearest fraction having a denominator smaller than *N*. To this end we can use the continued fraction expansion of *c/q* and Theorem 5.1.

We will introduce the following theorem which summarizes main results of the quantum algorithm for finding the order.

Theorem: If the integer *N* is sufficiently large then by repeating the first four steps of the algorithm for finding the order $O(\log\log N)$ times one can obtain the value of the order *r* with the probability $\gamma > 0$ where the constant γ does not depend on *N*.

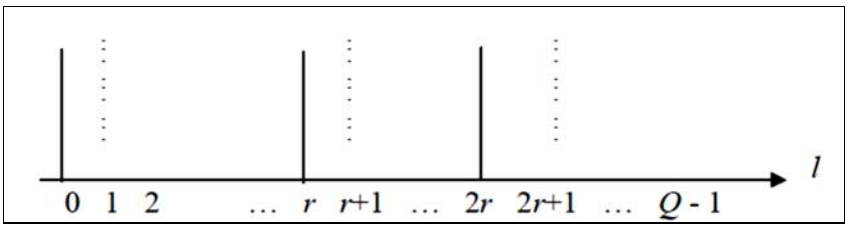
The quantum algorithm for order finding: Circuit work analysis

Again we will work with two registers. The first will hold a number between 1 to *Q*. (*Q* will be fixed later: it is much larger than *N*, but still polynomial in *N*.)The second register will carry numbers between 1 to *N*. Hence the two registers will consist of $O(\log(N))$ qubits.

Let us now understand how this algorithm works. In the second step of the algorithm, all numbers between 0 and $Q-1$ are present in the superposition, with equal weights. In the third step of the algorithm, they are separated to sets, each has periodicity *r*. This is done as follows: there are *r* possible values written on the second register: $a \in \left\{Y^0, Y^1, ...Y^{r-1}\right\}$. The third state can thus be written as:

$$\frac{1}{\sqrt{Q}}\left((\sum_{l=0|Y^l=Y}^{Q-1} |l\rangle \otimes |Y\rangle) + (\sum_{l=0|Y^l=Y^2}^{Q-1} |l\rangle \otimes |Y^2\rangle) + ... + (\sum_{l=0|Y^l=Y^r}^{Q-1} |l\rangle \otimes |Y^r = 1\rangle \right).$$

Note that the values *l* that give $Y^l = a$ have periodicity *r*: If the smallest such *l* is l_0, then $l = l_0 + r$, $l_0 + 2r$, … will also give $Y^l = a$. Hence each term in the brackets has periodicity *r*. Each set of *l's*, with periodicity *r*, is attached to a different state of the second register. Before the computation of Y^l, all *l's* appeared equally in the superposition. Writing down the Y^l on the second register can be thought of as giving a different "color" to each periodic set in [0, $Q-1$]. Visually, this can be viewed as follows:



The measurement of the second register picks randomly one of these sets, and the state collapses to a superposition of *l's* with periodicity *r*, with an arbitrary shift l_0. Now, how to obtain the periodicity? The first idea that comes to mind is to measure the first register twice, in order to get two samples from the same periodic set, and somehow deduce *r* from these samples.

However, the probability that the measurement of the second register yields the same shift in two runs of the algorithm, i.e. that the same periodic set is chosen twice, is exponentially small.

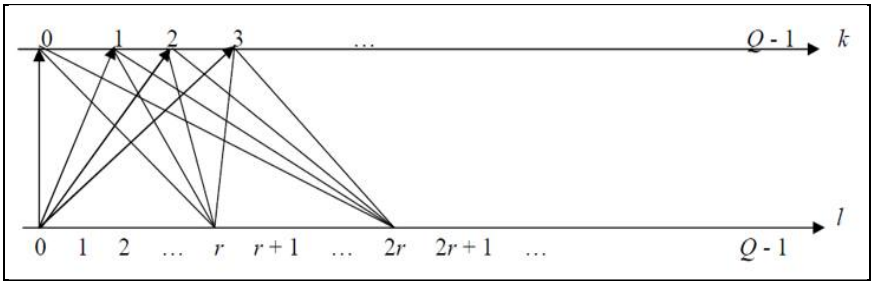
How to gain information about the periodicity in the state without simply sampling it?

This is done by the FT. To understand the operation of the FT, we use a diagram again:

Step	**Shor's algorithm**
1	$\left\|\vec{0}\right\rangle \otimes \left\|\vec{0}\right\rangle$
2	Apply FT over Z_Q on the first register $\frac{1}{\sqrt{Q}} \sum_{l=0}^{Q-1} \|l\rangle \otimes \left\|\vec{0}\right\rangle$
3	Call subroutine which computes $\|l\rangle\|d\rangle \mapsto \|l\rangle\|d \oplus Y^l \bmod N\rangle$ $\frac{1}{\sqrt{Q}} \sum_{l=0}^{Q-1} \|l\rangle \otimes \|Y^l \bmod N\rangle$
4	Measure second register $\frac{1}{\sqrt{A}} \sum_{l=0\|Y^l=Y^{l_0}}^{Q-1} \|l\rangle \otimes \|Y^{l_0}\rangle = \frac{1}{\sqrt{A}} \sum_{j=0}^{A-1} \|jr + l_0\rangle \otimes \|Y^{l_0}\rangle$

5	Apply FT over Z_Q on the first register $\frac{1}{\sqrt{Q}}\sum_{k=0}^{Q-1}\left(\frac{1}{\sqrt{A}}\sum_{j=0}^{A-1}e^{2\pi i(jr+l_0)k/Q}\right)\left\|k\right\rangle\otimes\left\|Y^{l_0}\right\rangle$
6	Measure first register.
7	Let k_1 be the outcome. Approximate the fraction $\frac{k_1}{Q}$ by a fraction with denominator smaller than *N*, using the (classical) method of continued fractions. If the denominator *d* doesn't satisfy $Y^d = 1 \bmod N$, throw it away. Else call the denominator r_1.
9	Repeat all previous steps poly(log(N)) times to get r_1, r_2, ..
10	Output the minimal *r*.

Each edge in the diagram indicates that there is some probability amplitude to transform from the bottom basis state to the upper one.



We now measure the first register, to obtain *k*. To find the probability to measure each *k*, we need to sum up the weights coming from all the *j's* in the periodic set.

$$\text{Prob}(k) = \left|\frac{1}{\sqrt{QA}}\sum_{j=0}^{A-1}e^{2\pi ik(jr+l_0)/Q}\right|^2 = \left|\frac{1}{\sqrt{QA}}\sum_{j=0}^{A-1}(e^{2\pi ikr/Q})^j\right|^2$$

Hence, in order to compute the probability to measure each *k*, we need to evaluate a geometrical series. Alternatively the geometric series is a sum over unit vectors in the complex plane.

Exact periodicity. Let us assume for a second *exact periodicity*, i.e. that *r* divides *Q* exactly. Then *A* = *Q*/*r*. In this case, the above geometrical series is equal to zero, unless $e^{2\pi ikr/Q} = 1$. Thus we measure with probability 1 only *k's* such that $kr = 0 \bmod Q$. This is where destructive interference comes to play: only "good" *k's*, which satisfy $kr = 0 \bmod Q$, remain, and all the others cancel out. Why are such *k's* "good" ? We can write *kr* = *mQ*, for some integer *m*, or *k*/*Q* = *m*/*r*. We know *k* since we have measured it. Therefore we can reduce the fraction *k*/*Q*. If *m* and *r* are coprime the denominator will be exactly *r* which we are looking for. The probability for all "good" *k's* is the same, so *m* is chosen randomly between 0 to *r* – 1. By the prime number theorem, there are approximately $r/\log(r)$ primes smaller than *r*. Repeating the experiment a large enough number of times we will with very high probability eventually get *m* prime, i.e. coprime to *r*.

Example. r divides q (easy case). Assume we have picked a random *x* and we want to find the corresponding period *r*. We can always efficiently pick some smooth *q* such that $N^2 < q \le 2N^2$ (for instance take *q* a power of 2). The QFT for Z_q can be implemented using $O((\log q)^2) = O((\log N)^2)$ elementary gates. We will first assume that the unknown *r* divides *q*, in which case everything works out smoothly. It is known that in $\tilde{O}((\log N)^2)$ steps we can compute the transformation $\left|a\right\rangle\left|0\right\rangle \to \left|a\right\rangle\left|x^a \bmod N\right\rangle$ using the Schonhage-

Strassen algorithm for fast multiplication. We now find r as follows. Start with $|0\rangle|0\rangle$, two registers of $\lceil \log q \rceil$ and $\lceil \log N \rceil$ zeroes, respectively. Apply the QFT to the first register to build $QFT := \frac{1}{\sqrt{q}} \sum_{a=0}^{q-1} |a\rangle|0\rangle$.

1 register	$\|O_1\rangle$	$\rightarrow$	$\lceil \log q \rceil$
2 register	$\|O_2\rangle$	$\rightarrow$	$\lceil \log N \rceil$

Then compute $x^a \bmod N$ in quantum parallel: $\frac{1}{\sqrt{q}} \sum_{a=0}^{q-1} |a\rangle |x^a \bmod N\rangle$. Observing the second register gives some $x^s \bmod N$, with $s < r$. Note that because r divides q, the a of the form $a = jr + s (0 \le j < q/r)$ are exactly the a for which $x^a \bmod N$ equals the observed value $x^s \bmod N$. Thus the first register collapses to a superposition of $|s\rangle, |r+s\rangle, |2r+s\rangle, ..., |q-r+s\rangle$ and the second register collapses to the classical state $x^s \bmod N$. We can now ignore the second register, and gave in the first: $\sqrt{\frac{r}{q}} \sum_{j=0}^{q/r-1} |jr+s\rangle$. Apply the QFT again gives

$$\sqrt{\frac{r}{q}} \sum_{j=0}^{q/r-1} \sum_{b=0}^{q-1} e^{2\pi i \frac{(jr+s)b}{q}} |b\rangle = \sqrt{\frac{r}{q}} \sum_{b=0}^{q-1} e^{2\pi i \frac{sb}{q}} \left(\sum_{j=0}^{q/r-1} e^{2\pi i \frac{jrb}{q}} \right) |b\rangle .$$

Using that $\sum_{j=0}^{n-1} a^j = (1-a^n)/(1-a)$ for $a \neq 1$, we compute:

$$\sum_{j=0}^{q/r-1} e^{2\pi i \frac{jrb}{q}} = \sum_{j=0}^{q/r-1} \left(e^{2\pi i \frac{rb}{q}} \right)^j = \begin{cases} q/r & \text{if } e^{2\pi i \frac{rb}{q}} = 1 \\ \dfrac{1 - \left(e^{2\pi i \frac{rb}{q}} \right)^{q/r}}{1 - e^{2\pi i \frac{rb}{q}}} = \dfrac{1 - e^{2\pi i b}}{1 - e^{2\pi i \frac{rb}{q}}} = 0 & \text{if } e^{2\pi i \frac{rb}{q}} \neq 1 \end{cases}$$

Note that $e^{2\pi i rb/q} = 1$ iff rb/q is an integer iff b is a multiple of q/r. Accordingly, we are left with a superposition where only the multiples of q/r have non-zero amplitude. Observing this final superposition gives some random multiple $b = cq/r$, with c a random number $0 \le c < r$. Thus we get a b such that $\frac{b}{q} = \frac{c}{r}$, where b and q are known and c and r are unknown. There are $\varphi(r) \in \Omega(r / \log\log r)$ numbers smaller than r which are coprime to r, so c will be coprime to r with probability $\Omega(1/\log\log r)$. Accordingly, an expected number of $O(\log\log N)$ repetitions of the procedure of this section suffices to obtain a $b = cq/r$ with c coprime to r. Once we have such a b, we can obtain r as the denominator by writing b/q in lowest terms.

Example. r does not divide q (hard case). In case r does not divide q (which is actually quite likely), it can be shown that applying exactly the same algorithm will still yield with high probability a b such that $\left| \frac{b}{q} - \frac{c}{r} \right| \le \frac{1}{2q}$, with b, q known and c, r unknown. Two distinct fractions, each with denominator $\le N$, must be at least $1/N^2 > 1/q$ apart.

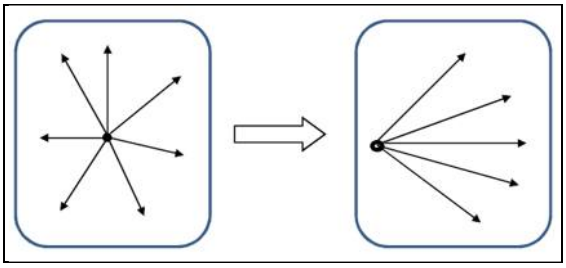
Remark. Consider two fractions $z = x/y$ and $z' = x'/y'$ with $y, y' \le N$. If $z \neq z'$ then $|xy' - x'y| \ge 1$, and hence $|z - z'| = |(xy' - x'y)/yy'| \ge 1/N^2$.

Therefore c/r is the only fraction with denominator $\le N$ at distance $\le 1/2q$ from b/q. Applying continued-fraction expansion to b/q efficiently gives us the fraction with denominator $\le N$ that is closest to b/q.

This fraction must be *c/r*. Again, with good probability *c* and *r* will be coprime, in which case writing *c/r* in lowest terms gives *r*. The whole algorithm finds a factor of *N* in expected time $\tilde{O}((\log N)^2)$.

Let us consider this case more in detail.

Example. Imperfect periodicity. In the general case, *r* does not divide *Q*, and this means that the picture is less clear. «Bad» *k's* do not completely cancel out. We distinguish between two types of *k's*, for which the geometrical series of vectors in the complex plain looks as follows:



In the left case, all vectors point in different directions, and they tend to cancel each other. This will cause destructive interference, which will cause the amplitude of such *k's* to be small. In the right case, all vectors point almost to the same direction. In this case there will be constructive interference of all the vectors. This happens when $e^{2\pi ikr/Q}$ is close to one, or when $kr \bmod Q$ is close to zero.

This means that with high probability, we will measure only *k's* which satisfy an *approximate* criterion $kr \approx 0 \bmod Q$. In particular, consider *k's* which satisfy: $-r/2 \le kr \bmod Q \le r/2$. There are exactly *r* values of *k* satisfying this requirement, because *k* runs from 0 to $Q-1$, therefore *kr* runs from 0 to $(Q-1)r$, and this set of integers contains exactly *r* multiples of *Q*.

Note, that for such *k's* all the complex vectors lie in the upper half of the complex plane, so they are instructively interfering. Now the probability to measure such a *k* is bounded below, by choosing the largest exponent possible:

Prob(*k*)	=	$\left\|\frac{1}{\sqrt{QA}}\sum_{j=0}^{A-1}(e^{2\pi ikr/Q})^j\right\|^2 \ge \left\|\frac{1}{\sqrt{QA}}\sum_{j=0}^{A-1}(e^{i\pi r/Q})^j\right\|^2$				
	=	$\frac{1}{QA}\left\|\frac{1-e^{\pi irA/Q}}{1-e^{i\pi r/Q}}\right\|^2$	=	$\frac{1}{QA}\left\|\frac{\sin\left(\frac{\pi rA}{2Q}\right)}{\sin\left(\frac{\pi r}{2Q}\right)}\right\|^2$	≈	$\frac{4}{\pi^2 r}$

Quantum algorithm to calculate the order

Let us any practical examples of Shor's algorithm application.

Example. Quantum circuit for finding the order of the positive integer x mod *N*. Consider the circuit of Fig. 12. It calculates the order *r* of the positive integer x less than *N*, coprime to *N*.

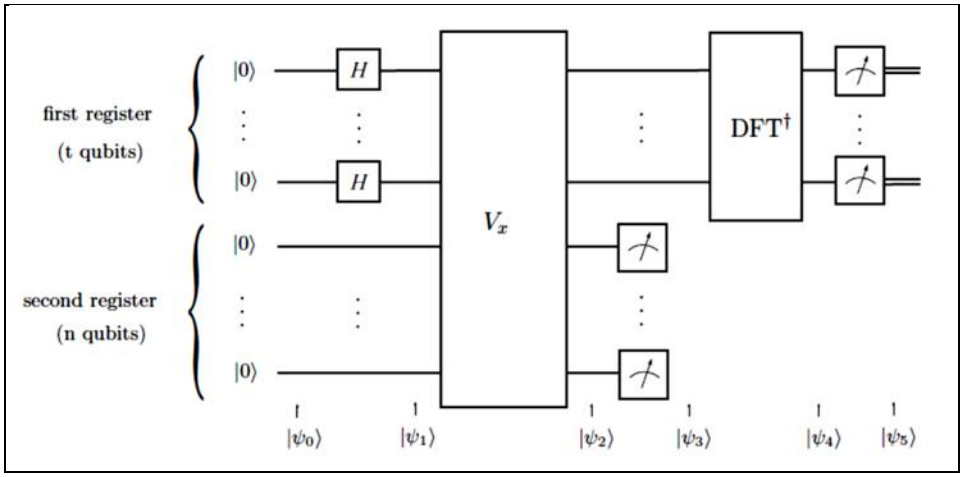


Figure 12. Quantum circuit for finding the order of the positive integer x mod N

V_x is the unitary linear operator

$$V_x\left(|j\rangle|k\rangle\right)=|j\rangle\left|k+x^j\right\rangle \tag{4}$$

where $|j\rangle$ and $|k\rangle$ are the states of the first and second registers, respectively. The arithmetical operations are performed mod N, so $0 \le k + x^j < N$. DFT is the Discrete Fourier Transform operator which will be described ahead.

The first register has t qubits, where t is generally chosen such that $N^2 \le 2^t < 2N^2$, for reasons that will become clear later. As an exception, if the order r is a power of 2, then it is enough to take $t = n$. In this section we consider this very special case and leave the general case for next section. We will keep the variable t in order to generalize the discussion later on.

The states of the quantum computer are indicated by $|\psi_0\rangle$ to $|\psi_5\rangle$ in Fig. 12. The initial state is

$$|\psi_0\rangle=\underbrace{|0\ldots0\rangle}_{t}\underbrace{|0\ldots0\rangle}_{n}.$$

The application of the Hadamard operator on each qubit of the first register yields

$$|\psi_1\rangle=\frac{1}{\sqrt{2^t}}\sum_{j=0}^{2^t-1}|j\rangle|0\rangle. \tag{5}$$

The first register is then in a superposition of all states of the computational basis with equal amplitudes given by $\frac{1}{\sqrt{2^t}}$. Now we call the reader's attention to what happens when we apply V_x to $|\psi_1\rangle$ as:

$$|\psi_2\rangle=V_x|\psi_1\rangle=\frac{1}{\sqrt{2^t}}\sum_{j=0}^{2^t-1}V_x\left(|j\rangle|0\rangle\right)=\frac{1}{\sqrt{2^t}}\sum_{j=0}^{2^t-1}|j\rangle\left|x^j\right\rangle. \tag{6}$$

The state $|\psi_2\rangle$ is a remarkable one. Because V_x is linear, it acts on all $|j\rangle|0\rangle$ for 2^t values of j, so this generates all powers of x simultaneously. This feature is called quantum parallelism. Some of these powers are 1, which correspond to the states $|0\rangle|1\rangle,|r\rangle|1\rangle,|2r\rangle|1\rangle,\ldots,\left|\left(\frac{2^t}{r}-1\right)r\right\rangle|1\rangle$. This explains the choice (4) for V_x.

Classically, one would calculate successively x^j, for j starting from 2 until reaching $j = r$.

Quantumly, one can calculate all powers of x with just one application of V_x.

At the quantum level, the values of j that yield $x_j \equiv 1 \bmod N$ are "known". But this quantum information is not fully available at the classical level. Classical information of a quantum state is obtained by practical

measurements and, at this point, it does not help if we measure the first register, since all states in the superposition (1.10) have equal amplitudes. The first part of the strategy to find *r* is to observe that the first register of the states $|0\rangle|1\rangle,|r\rangle|1\rangle,|2r\rangle|1\rangle,\ldots,|2^t-r\rangle|1\rangle$ is periodic. So the information we want is a period. In order to simplify the calculation, let us measure the second register. Before doing this, we will rewrite $|\psi_2\rangle$ collecting equal terms in the second register. Since x_j is a periodic function with period *r*, substitute *ar* + *b* for *j* in Eq. (1.10), where $0 \le a \le (2^t/r) - 1$ and $0 \le b \le r - 1$. Recall that we are supposing that $t = n$ and *r* is a power of 2, therefore *r* divides 2^t. Eq. (6) is converted to

$$|\psi_2\rangle = \frac{1}{\sqrt{2^t}} \sum_{b=0}^{r-1} \left(\sum_{a=0}^{\frac{2^t}{r}-1} |ar+b\rangle \right) |x^b\rangle \tag{7}$$

In the second register, we have substituted x^b for x^{ar+b}, since $x^r \equiv 1 \bmod N$. Now the second register is measured. Any output $x^0, x^1, \ldots, x^{r-1}$ can be obtained with equal probability. Suppose that the result is x^{b_0}. The state of the quantum computer is now

$$|\psi_3\rangle = \sqrt{\frac{r}{2^t}} \left(\sum_{a=0}^{\frac{2^t}{r}-1} |ar+b_0\rangle \right) |x^{b_0}\rangle \quad . \tag{8}$$

Remark. Note that after the measurement, the constant is renormalized to $\sqrt{r/2^t}$, since there are $2^t / t$ terms in the sum (8). Fig. 13 shows the probability of obtaining the states of the computational basis upon measuring the first register.

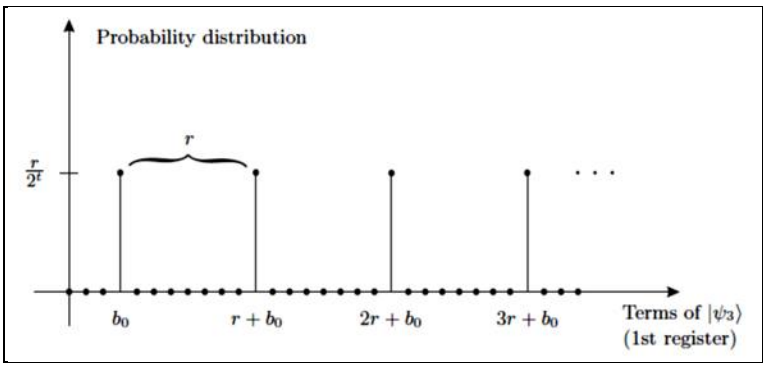


Figure 13. Probability distribution of $|\psi_3\rangle$ measured in the computational basis

(for the case $b_0 = 3$ and $r = 8$)

The horizontal axis has 2^t points. The number of peaks is 2^t/r and the period is *r*. The probabilities form a periodic function with period *r*. Their values are zero except for the states $|b_0\rangle,|r+b_0\rangle,|2r+b_0\rangle,\ldots,|2^t-r+b_0\rangle$.

How can one find out the period of a function efficiently? The answer is in the Fourier transform (FT). The FT of a periodic function with period *r* is a new periodic function with period proportional to 1/*r*. This makes a difference for finding *r*.

The FT is the second and last part of the strategy. The whole method relies on an efficient QA for calculating the FT, which is not available classically. In next section, we show that the FT is calculated efficiently in a quantum computer.

Example. The quantum discrete FT (DFT). The FT of the function $F:\{0,\ldots,N-1\} \to \mathbb{C}$ is a new function $\tilde{F}:\{0,\ldots,N-1\} \to \mathbb{C}$ defined as

$$\tilde{F}(k)=\frac{1}{\sqrt{N}}\sum_{j=0}^{N-1}e^{2\pi ijk/N}F(j). \tag{9}$$

The FT can be applied either on a function or on the states of the computational basis. The FT applied to the state $|k\rangle$ of the computational basis $\{|0\rangle,\ldots,|N-1\rangle\}$ is

$$DFT(|k\rangle)=|\psi_k\rangle=\frac{1}{\sqrt{N}}\sum_{j=0}^{N-1}e^{2\pi ijk/N}|j\rangle \tag{10}$$

where the set $\{|\psi_k\rangle : k=0,\ldots,N-1\}$ forms a new orthonormal basis, i.e., $\langle\psi_{k'}|\psi_k\rangle=\delta_{k'k}$. The FT is a unitary linear operator. So, if we know how it acts on the states of the computational basis, we also know how it acts on a generic state: $|\psi\rangle=\sum_{a=0}^{N-1}F(a)|a\rangle$.

The FT of $|\psi\rangle$ can be performed indistinctly using either (9) or (10).

Now we will continue the calculation process of the circuit of Fig. 12. We are ready to find out the next state of the quantum computer — $|\psi_4\rangle$. Applying the inverse FT on the first register, using Eq. (10) and the linearity of DFT†, we obtain

$$|\psi_4\rangle=DFT^{\dagger}(|\psi_3\rangle)=\sqrt{\frac{r}{2^t}}\sum_{a=0}^{\frac{2^t}{r}-1}\left(\frac{1}{\sqrt{2^t}}\sum_{j=0}^{\frac{2^t}{r}-1}\left(e^{-2\pi ij(ar+b_0)/2^t}\right)|j\rangle\right)|x^{b_0}\rangle.$$

Inverting the summation order, we have

$$|\psi_4\rangle=\frac{1}{\sqrt{r}}\left(\sum_{j=0}^{2^t-1}\left[\frac{r}{2^t}\sum_{a=0}^{\frac{2^t}{r}-1}\left(e^{-\frac{2\pi ija}{2^t}r}\right)\right]e^{-\frac{2\pi ijb_0}{2^t}}|j\rangle\right)|x^{b_0}\rangle \tag{11}$$

Using (10), we see that the expression in square brackets is not zero if and only if $j = k2^t/r$, with $k = 0,\ldots, r-1$. When j takes such values, the expression in the square brackets is equal to 1. So we have

$$|\psi_4\rangle=\frac{1}{\sqrt{r}}\left(\sum_{k=0}^{r-1}e^{-2\pi i\frac{k}{r}b_0}\left|\frac{k2^t}{r}\right\rangle\right)|x^{b_0}\rangle \tag{12}$$

.

In order to find r, the expression for $|\psi_4\rangle$ has two advantages over the expression for $|\psi_3\rangle$ (Eq. (1.12)): r is in the denominator of the ket label and the random parameter b_0 moved from the ket label to the exponent occupying now a harmless place.

Fig. 13 shows the probability distribution of $|\psi_4\rangle$ measured in the computational basis. Measuring the first register, we get the value $k_0 2^t/r$, where k_0 can be any number between 0 and $r-1$ with equal probability (the peaks in Fig. 1.32). If we obtain $k_0 = 0$, we have no clue at all about r, and the algorithm must be run again. If $k_0 \neq 0$, we divide $k_0 2^t/r$ by 2^t, obtaining k_0/r. Neither k_0 nor r are known. If k_0 is coprime to r, we simply select the denominator.

If k_0 and r have a common factor, the denominator of the reduced fraction k_0/r is a factor of r but not r itself. Suppose that the denominator is r_1. Let $r = r_1 r_2$. Now the goal is to find r_2, which is the order of x^{r_1}. We run again the quantum part of the algorithm to find the order of x^{r_1}. If we find r_2 in the first round, the algorithm halts, otherwise we apply it recursively. The recursive process does not last, because the number of iterations is less than or equal to $\log_2 r$.

Take $N = 15$ as an example, which is the least nontrivial composite number.

The set of numbers less than 15, coprime to 15 is {1, 2, 4, 7, 8, 11, 13, 14}. The numbers in the set {4, 11, 14} have order 2 and the numbers in the set {2, 7, 8, 13} have order 4. Therefore, in any case r is a power

of 2 and the factors of $N = 15$ can be found in a 8-bit quantum computer $\left(t + n = 2\lceil \log_2 15 \rceil = 8\right)$. A 7-qubit quantum computer is used, bypassing part of the algorithm.

We have considered a special case when the order r is a power of 2 and $t = n$ (t is the number of qubits in the first register — Fig. 1.31 — and $n = \lceil \log_2 N \rceil$).

Now we consider the factorization of $N = 21$, that is the next nontrivial composite number.

Example. Generalization by means. We must choose t such that 2^t is between N^2 and $2N^2$, which is always possible. For $N = 21$, the smallest value of t is 9. This is the simplest example allowed by the constraints, but enough to display all properties of Shor's algorithm.

The first step is to pick up x at random such that $1 < x < N$, and to test whether x is coprime to N. If not, we easily find a factor of N by calculating *gcd* (x, N). If yes, the quantum part of the algorithm starts. Suppose that $x = 2$ has been chosen. The goal is to find out that the order of x is $r = 6$. The quantum computer is initialized in the state $|\psi_0\rangle = |0\rangle|0\rangle$, where the first register has $t = 9$ qubits and the second has $n = 5$ qubits.

Next step is the application of $H^{\otimes 9}$ on the first register yielding (see Eq. (5))

$$|\psi_1\rangle = \frac{1}{\sqrt{512}} \sum_{j=0}^{511} |j\rangle|0\rangle .$$

The next step is the application of V_x (defined in (6)), which yields

$$|\psi_2\rangle = \frac{1}{\sqrt{512}} \sum_{j=0}^{511} |j\rangle \left|2^j \bmod N\right\rangle =$$

$$= \frac{1}{\sqrt{512}} \begin{pmatrix} |0\rangle|1\rangle + |1\rangle|2\rangle + |2\rangle|4\rangle + |3\rangle|8\rangle + |4\rangle|16\rangle + |5\rangle|11\rangle + \\ |6\rangle|1\rangle + |7\rangle|2\rangle + |8\rangle|4\rangle + |9\rangle|8\rangle + |10\rangle|16\rangle + |11\rangle|11\rangle + \\ |12\rangle|1\rangle + \ldots \end{pmatrix}$$

Notice that the above expression has the following pattern: the states of the second register of each «column» are the same.

Therefore we can rearrange the terms in order to collect the second register:

$$|\psi_2\rangle =$$

$$= \frac{1}{\sqrt{512}} \begin{pmatrix} \left(|0\rangle + |6\rangle + |12\rangle + \ldots + |504\rangle + |510\rangle\right)|1\rangle + \left(|1\rangle + |7\rangle + |13\rangle + \ldots + |505\rangle + |511\rangle\right)|2\rangle + \\ \left(|2\rangle + |8\rangle + |14\rangle + \ldots + |506\rangle\right)|4\rangle + \left(|3\rangle + |9\rangle + |15\rangle + \ldots + |507\rangle\right)|8\rangle + \\ \left(|4\rangle + |10\rangle + |16\rangle + \ldots + |508\rangle\right)|16\rangle + \left(|5\rangle + |11\rangle + |17\rangle + \ldots + |509\rangle\right)|11\rangle \end{pmatrix}. \qquad (13)$$

This feature was made explicit in Eq. (7). Because the order is not a power of 2, here there is a small difference: the first two lines of Eq. (13) have 86 terms, while the remaining ones have 85.

Now one measures the second register1, yielding one of the following numbers equiprobably: {1, 2, 4, 8, 16, 11}. Suppose that the result of the measurement is 2, then

$$|\psi_3\rangle = \frac{1}{\sqrt{86}} \left(|1\rangle + |7\rangle + |13\rangle + \ldots + |505\rangle + |511\rangle\right)|2\rangle \ . \qquad (14)$$

Notice that the state $|\psi_3\rangle$ was renormalized in order to have unit norm. It does not matter what is the result of the measurement; what matters is the periodic pattern of (14).

The period of the states of the first register is the solution to the problem and the FT can reveal the value of the period. So, the next step is the application of the inverse Fourier transform on the first register of $|\psi_3\rangle$:

$$\begin{aligned}\left|\psi_4\right\rangle &= DFT^{\dagger}\left(\left|\psi_3\right\rangle\right) = DFT^{\dagger}\left(\frac{1}{\sqrt{86}}\sum_{a=0}^{85}\left|6a+1\right\rangle\right)\left|2\right\rangle \\ &= \frac{1}{\sqrt{512}}\sum_{j=0}^{511}\left(\left[\frac{1}{\sqrt{86}}\sum_{a=0}^{85}e^{-2\pi i\frac{6ja}{512}}\right]e^{-2\pi i\frac{j}{512}}\left|j\right\rangle\right)\left|2\right\rangle\end{aligned} \tag{15}$$

where we have used Eq. (10) and have rearranged the sums. The last equation is similar to Eq. (11), but with an important difference. We were assuming that r divides 2^t. This is not true in the present example (6 does not divide 512), therefore we cannot use the identity to simplify the term in brackets in Eq. (11). This term never vanishes, but its main contribution is still around j = 0, 85, 171, 256, 341, 427, which are obtained rounding $512k_0/6$ for k_0 from 0 to 5.

To see this, let us plot the probability of getting the result j (in the interval 0 to 511) by measuring the first register of the state $\left|\psi_4\right\rangle$. From (15), we have that the probability is

$$\text{Prob}(j) = \frac{1}{512\times 86}\left|\sum e^{-2\pi i\frac{6ja}{512}}\right|^2 \tag{1.20}$$

The plot of Prob (j) is shown in Fig. 14.

We see the peaks around j = 0, 85, 171, 256, 341, 427, indicating a high probability of getting one of these values, or some value very close to them. In between, the probability is almost zero. The sharpness of the peaks depends on t (number of qubits in the first register). The lower limit $2^t \geq N^2$ ensures a high probability in measuring a value of j carrying the desired information.

Let us analyze the possible measurement results. If we get j = 0 (first peak), the algorithm has failed in this round. It must be run again. We keep x = 2 and rerun the quantum part of the algorithm. The probability of getting j = 0 is low: from Eq. (1.24) we have that Prob(0) = 86/512 ≈ 0.167. Now suppose we get j = 85 (or any value in the second peak). We divide by 512 yielding 85/512, which is a rational approximation of $k_0/6$, for k_0 = 1.

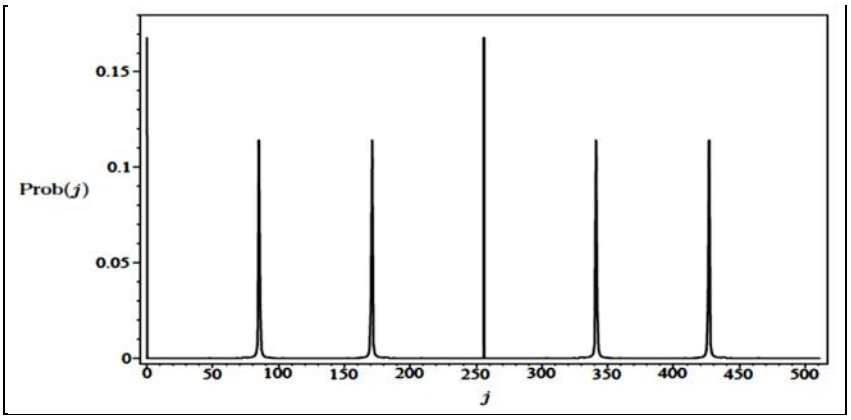


Figure 14. Plot of Prob(j) against j

Compare to the plot of Fig. 13, where peaks are not spread and have the same height

How can we obtain r from 85/512? The method of continued fraction approximation allows one to extract the desired information. A general continued fraction expansion of a rational number j_1/j_2 has the form

$$\frac{j_1}{j_2} = a_0 + \cfrac{1}{a_1 + \cfrac{1}{\ldots + \cfrac{1}{a_p}}},$$

usually represented as $[a_0, a_1, ..., a_p]$, where a_0 is a non-negative integer and a_1, ..., a_p are positive integers. The q-th convergent ($0 \leq q \leq p$) is defined as the rational number $[a_0, a_1, ..., a_q]$. It is an approximation to j_1/j_2 and has a denominator smaller than j_2. This method is easily applied by inversion of the fraction followed by

integer division with rational remainder. Inverting 85/512 yields 512/85, which is equal to 6+2/85. We repeat the process with 2/85 until we get numerator 1. The result is

$$\frac{85}{512} = \frac{1}{6 + \dfrac{1}{42 + \dfrac{1}{2}}}.$$

So, the convergents of 85/512 are 1/6, 42/253, and 85/512. We must select the convergents that have a denominator smaller than $N = 21$ (since $r < N$). This method yields 1/6, and then $r = 6$. We check that $2^6 \equiv 1 \mod 21$, and the quantum part of the algorithm ends with the correct answer. The order $r = 6$ is an even number, therefore $gcd(2^{(6/2)} \pm 1, 21)$ gives two non trivial factors of 21. A straightforward calculation shows that any measured result in the second peak (say $81 \le j \le 89$) yields the convergent 1/6.

Consider now the third peak, which corresponds to $k_0/6$, $k_0 = 2$. We apply again the method of continued fraction approximation, which yields 1/3, for any j in the third peak (say $167 \le j \le 175$). In this case, we have obtained a factor of r ($r_1 = 3$), since $2^3 \equiv 8\ 6 \equiv 1 \mod 21$. We run the quantum part of the algorithm again to find the order of 8. We eventually obtain $r_2 = 2$, which yields $r = r_1 r_2 = 3 \times 2 = 6$. The fourth and fifth peaks yield also factors of r. The last peak is similar to the second, yielding r directly.

The general account of the succeeding probability is as follows. The area under all peaks is approximately the same: ≈ 0.167. The first and fourth peaks have a nature different from the others — they are not spread. To calculate their contribution to the total probability, we take the basis equal to 1. The area under the second, third, fifth, and last peaks are calculated by adding up Prob(j), for j running around the center of each peak.

So, in approximately 17% cases, the algorithm fails (1st peak). In approximately 33% cases, the algorithm returns r in the first round (2nd and 6th peaks). In approximately 50% cases, the algorithm returns r in the second round or more (3rd, 4th, and 5th peaks).

Now we calculate the probability of finding r in the second round. For the 3rd and 5th peaks, the remaining factor is $r_2 = 2$. The graph equivalent to Fig. 14 in this case has 2 peaks, then the algorithm returns r_2 in 50% cases. For the 4th peak, the remaining factor is $r = 3$ and the algorithm returns r_2 in 66.6% cases. This amounts to $\dfrac{2 \times 50\% + 66.6\%}{3}$ of 50%, which is equal to around 22%. In summary, the success probability for $x = 2$ is around 55%.

Remark. We have shown that Shor's algorithm is an efficient probabilistic algorithm, assuming that the FT could be implemented efficiently. The complete circuit for the QFT is given in Fig. 15.

Now we can calculate the complexity of the quantum Fourier circuit. Counting the number of elementary gates we get the leading term $5n^2/2$, which implies that the complexity is $O(n^2)$.

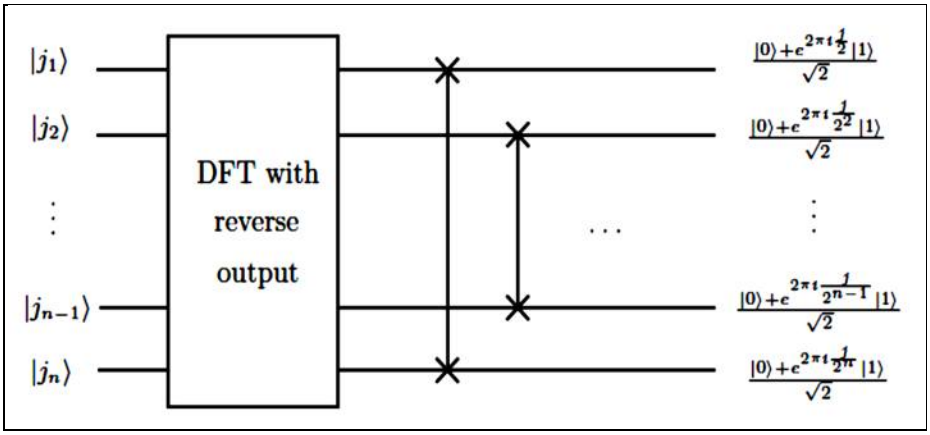


Figure 15. The complete circuit for the quantum Fourier Transform

By now one should be asking about the decomposition of V_x in terms of the elementary Fourier Transform. V_x is the largest gate of Fig. 12. Actually, Shor stated in his 1997 paper that V_x is the «bottleneck of the

quantum factoring algorithm» due to the time and space consumed to perform the modular exponentiation. The bottleneck is not so strict though since, by using the well known classical method of repeated squaring and ordinary multiplication algorithms, the complexity to calculate modular exponentiation is $O(n^3)$. The quantum circuit can be obtained from the classical circuit by replacing the irreversible classical gates by the reversible quantum counterpart. *Vx* is a problem in recursive calls of the algorithm when *x* changes. For each *x*, a new circuit must be built, what is troublesome at the present stage of hardware development.

References

1. Lo H.-K., Popescu S. and Spiller T. (Eds). Introduction to quantum computing and information. – World Scientific Publ. Co. – 1998.
2. Gruska J. Quantum computing // Advanced Topics in Computer Science Series, McGraw-Hill Companies. – London, 1999.
3. Pittenberg A.O. An introduction to quantum computing and algorithms. – Progress in Computer Sciences and Applied Logic. – Birkhauser. – 1999. – Vol. 19.
4. Berman G.P., Doolen G.D., Mainieri R. and Tsifrinovich V.I. Introduction to quantum computers // World Scientific Publ. Co. – 1999.
5. Ulyanov S.V., Ghisi F., Kurawaki I. and Litvintseva L.V. Simulation of quantum algorithms on classical computer. – Note del Polo Ricerca, Università degli Studi di Milano (Polo Didattico e di Ricerca di Crema). – Vol. 32. – Milan, 1999.
6. Nielsen M.A. and Chuang I.L. Quantum Computation and Quantum Information. – UK: Cambridge Univ. Press, 2000.
7. Hirvensalo M. Quantum computing // Natural Computing Series, Springer-Verlag, Berlin, 2001.
8. Hardy Y. and Steeb W.-H. Classical and quantum computing with C++ and Java Simulations. – Birkhauser Verlag, Basel, 2001.
9. Calude C.S. and Paun G. Computing with cells and atoms: An introduction to quantum, DNA and membrane computing. – N.Y.: Taylor&Francis, 2001.
10. Kitaev A.Yu., Shen A.H., Vyaly M.N. Classical and quantum computation. – N.Y.: AMS, 2002.
11. Brylinski F.K. and Chen G. (Eds). Mathematics of quantum computation. – Computational Mathematics Series. – CRC Press Co, 2002.
12. Ulyanov S.V., Litvintseva L.V., Ulyanov I.S. and Ulyanov S.S. Quantum information and quantum computational intelligence: Quantum decision making and search algorithms. – Note del Polo Ricerca, Università degli Studi di Milano (Polo Didattico e di Ricerca di Crema). – Milan, 2005. – Vol. 84-85.
13. Stenholm S. and Suominen K.-A. Quantum approach to informatics. – Wiley- Interscience. J. Wiley&Sons, Inc. – 2005.
14. Marinescu D.C. and Marinescu G.M. Approaching quantum computing. – Pearson Prentice Hall, New Jersey. – 2005.
15. Benenti G., Casati G., Strini G. Principles of quantum computation and information. – Singapore: World Scientific. –2004, Vol. I.; – 2007. – Vol. II.
16. Janzing D. Computer science approach to quantum control. – Habilitation: Univ. Karlsruhe (TH) Publ. Germany. – 2006.
17. Jaeger G. Quantum Information: An Overview. – N.Y.: Springer Verlag, 2007.
18. Kaye P., Laflamme R. and Mosca M. An introduction to quantum computing. – N.Y.: Oxford University Press, 2007.
19. McMahon D. Quantum computing explained // Wiley Interscience. A J. Wiley Sons, Inc. – 2008.
20. Lanzagorta M. and Uhlmann J. Quantum computer science // Morgan & Claypool Publ. – Series: SYNTHESIS LECTURES ON QUANTUM COMPUTING (Lecture #2). – 2009.

21. Nakahara M. and Ohmi T. Quantum computing: From Linear Algebra to Physical Realizations // Taylor & Francis. – 2008.
22. Chen G., Kauffman L., and Lomonaco S. J. Mathematics of Quantum Computation and Quantum Technology – N.Y.: Chapman Hall/CRC (Applied Mathematics and Nonlinear Science Series), 2008.
23. Chen G., Church D.A., Englert B.-G., Henkel C., Rohwedder B., Scully M.O. and Zubairy M.S. Quantum Computing Devices: Principles, Designs, and Analysis. – N.Y.: Chapman Hall/CRC (Applied Mathematics and Nonlinear Science Series), 2008.
24. McMahon D. Quantum computing explained. – N.J.: John Wiley & Sons. – 2008.
25. Yanofsky N.S. and Mannucci M.A. Quantum Computing for Computer Scientists. – Cambridge University Press. – 2008.
26. Chen G. and Diao. Mathematical Theory of Quantum Computation. – N.Y.: Chapman Hall/CRC (Applied Mathematics and Nonlinear Science Series), 2009.
27. Kholevo A.S. Quantum systems, channels, and information. – M.: МЦНМО, 2010 (in Russian).
28. Lavor C., Manssur L.R.U. and Portugal R. Grover's algorithm: Quantum database search // arXiv:quant-ph/0301079v1 16 Jan 2003.
29. Lomonaco S.J. (Jr) A lecture on Grover's quantum search algorithm (Version 1.1) // arXiv: quant-ph/0010040v2 18 Oct 2000.
30. Lavor C., Manssur L.R.U. and Portugal R. Shor's algorithm for factoring large integers // arXiv: quant-ph/0303175v1 29 Mar 2003.
31. Galindo A. and Martin-Delgado M.A. Information and computation: Classical and quantum aspects // Review of Modern Physics. – 2002. – Vol. 74. – № 2. – Pp. 347-423.
32. Batty M., Braunstein S.L., Duncan A.J. and Rees S. Quantum algorithms in group theory // arXiv: quant-ph/0310133v1, 21 Oct 2003. – P. 52.
33. Quantum Algorithms: Shor's algorithm, Grover's algorithm, Quantum logic, Quantum algorithm, Quantum Fourier transform, Deutsch-Jozsa algorithms. – Books LLC. – 2010.
34. Rieffel E. G. and Polak W. H. Quantum Computing: A gentle introduction. – B.: The MIT Press. – 2011.
35. Ohya M. and Volovich I. Mathematical foundations of quantum information and computation and its applications to nano- and bio-systems. – N.Y.: Springer Verlag, 2011.