Quantum Algorithms

Seyed Massoud Amini
Department of Mathematics
Tarbiat Modares University
Tehran 14115-175, Iran

Laboratory of Theoretical Studies
Institute for Mathematical Research
Universiti Putra Malaysia

mamini@modares.ac.ir, massoud@putra.upm.edu.my

Abstract
We give a short overview of quantum algorithms. Some famous algorithms such as Deutsch-Jozsa and Simon are covered with more details. The directions for further development are addressed.

Introduction

Using quantum-mechanical properties of natural particles, it is possible to build computers that are different in some interesting ways from the ones that we are used to.

In our usual computers, a bit is either 0 or 1 at a particular time. Hence a group (register) of n bits can contain only one of 2^n different numbers at a given time. A quantum bit (*qubit*) can be in a weighted combination (*superposition*) of *both* 0 and 1 at the same time. A group of n qubits can therefore hold all of those 2^n different numbers simultaneously.

When we measure a qubit, a given superposition boils down to a position. In general, this is a probabilistic event, and the probabilities are determined by the state of the quantum bit before the measurement. To represent the exact state of a qubit, we have to specify two complex numbers (*amplitudes*) which describe the particular combination of 0 and 1 in that state.

To distinguish qubits from "classical" bits, it is common to use the *bra-ket* notation of quantum mechanics. So the expression $|0\rangle$ represents quantum zero, and $|1\rangle$ represents quantum one. The left-sided notation $\langle 0|$ would be used if one deals with co-vectors. The state of a qubit can then be represented by $\alpha|0\rangle+\beta|1\rangle$, where α and β are the amplitudes of $|0\rangle$ and $|1\rangle$, respectively, in this state. When we measure this qubit, we see $|0\rangle$ with probability $|\alpha|^2$, and $|1\rangle$ with probability $|\beta|^2$. The laws of physics say that, in any quantum register, the sum of these square terms must be 1. So in our example, $|\alpha|^2+|\beta|^2=1$. Similarly, an n-qubit register will have 2^n amplitudes, each determining the probability that the binary number corresponding to it will be seen when the register is read, and the sum of squares of their absolute values will be 1.

We show the group of amplitudes that describe the state of a quantum register as a vector. A qubit with state $|0\rangle$, which is guaranteed to read 0 when it is measured, is represented by the vector $\begin{pmatrix}1\\0\end{pmatrix}$, and a qubit with state $|1\rangle$ is represented by the vector $\begin{pmatrix}0\\1\end{pmatrix}$. These two vectors are called the *basis states*. So the amplitude of the value $|0\rangle$ is given in the first row of the vector, and the amplitude of the value $|1\rangle$ is given in the second row. We assume that we have the machinery to set any qubit to one of these two values at the beginning of our algorithm. An arbitrary qubit state is then represented by the vector $\begin{pmatrix}\alpha\\\beta\end{pmatrix}$, where α and β are complex numbers and $|\alpha|^2+|\beta|^2=1$.

This can be generalized to n qubits easily. The state of a register consisting of n qubits is represented by the tensor product (⊗) of the individual states of the qubits in it.

A quantum program which transforms a register of n qubits from an input state to an output state can be represented as a 2^n by 2^n matrix, which, when multiplied by the input state vector, gives the output state vector. The laws of physics say that this matrix should be a unitary matrix. A unitary matrix is a square matrix M of complex numbers which has the following property: Replace each member $a+b.i$ of M with the number $a-b.i$. Take the transpose of this matrix. Multiply this new matrix (called $M^\dagger$) with the original matrix M. The result should be the identity matrix. This property guarantees that every quantum program is reversible, i.e., the inverse of the function that it computes can also be computed by a quantum program to obtain the original input from the output.

To code a "classical" algorithm in the quantum format, we have to make sure that our program never deletes information. Any classical program can be modified to ensure this, and for every classical program which computes an output from a given input, we can construct a quantum program (i.e. we can write the matrix mentioned above) which performs the same transformation on a register big enough to hold both the input and the output.

Consider the following simple quantum program working on a single qubit:

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

This is called an Hadamard gate. When the input is $|0\rangle$, this program changes the state of the qubit to

$$\begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix},$$

that is, to $\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$. So when you read the qubit at the end, you have exactly 50% chance of seeing a 0, and an equal chance of seeing a 1. Classical computers can never create truly random numbers, but this program can.

The combined effect of several single-qubit programs which run parallel on the qubits in a register can be represented as the tensor product of their individual matrices. So if we apply the H program to each qubit of a 2-qubit register, the result of this procedure can be calculated by multiplying the matrix

$$\begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix} \otimes \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix} = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & -\frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \end{pmatrix}$$

with the 4-element amplitude vector describing the initial state of the two qubits. We can generalize this to n qubits easily. So when you have an n-qubit register, the effect of applying H to each qubit in parallel can be computed by multiplying the matrix that you would obtain by taking the tensor product of n H's with the 2^n-element vector describing the initial state of the register. Generalizing the example above, if this n-qubit register originally contains the value $|0^n\rangle$, it is transformed to the "superposition" state

$$\frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle$$

by this program, and we would see each of the 2^n binary numbers x with equal probability when we observe the register.

When the input to a single Hadamard gate is $|1\rangle$, we see that the output is

$$\begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{pmatrix},$$

or, in our alternative notation, $\frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle$.

Let's see what the parallel application of H gates do to a 2-qubit register originally containing $|01\rangle$:

$$\begin{pmatrix} \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & -\frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} \frac{1}{2} \\ -\frac{1}{2} \\ \frac{1}{2} \\ -\frac{1}{2} \end{pmatrix},$$

or $\frac{1}{2}|0\ \rangle - \frac{1}{2}|0\ \rangle + \frac{1}{2}|0\ \rangle - \frac{1}{2}|1\ \rangle$.

As these examples indicate, whenever we apply the H gate in parallel to a register initially at an arbitrary basis state $|x\rangle$, where x is a binary number with n bits, the output state is a superposition of all the different 2^n basis states, where the absolute value of the amplitude of each basis state is $\frac{1}{\sqrt{2^n}}$, and the sign of the amplitude of such a basis state $|z\rangle$ in the resulting superposition is positive if an even number of bits which were 1 in the original state $|x\rangle$ are still 1 in $|z\rangle$, and negative otherwise. In other words, the parallel application of H gates to all qubits of an n-qubit register initially at state $|x\rangle$ results in the state

$$\frac{1}{\sqrt{2^n}} \sum_{z=0}^{2^n-1} (-1)^{x \cdot z} |z\rangle$$

where $x \cdot z = \sum_{i=1}^{n} x_i . z_i$, such that x_i and z_i are the ith bit of x and z, respectively.

The Quantum Circuit Model

A classical computer can be described by a circuit. The input is a string of bits. The input is processed by a succession of logical gates like NOT, OR, AND or NAND, which transform the input to the output. In general the output bits are Boolean functions of the input bits. A schematic view is depicted in

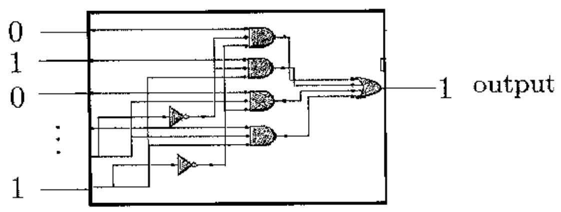
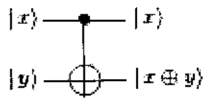


Figure 1: A classical circuit computing a Boolean function.

In the quantum model we use quantum gates. Here is an example.

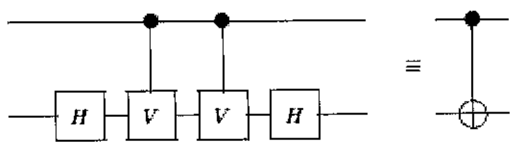
$$C = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$



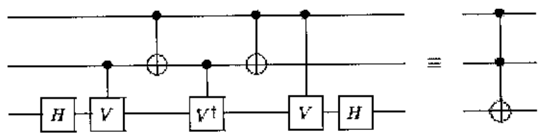
The idea is using unitary matrices as gates, such as

$$V = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}$$

The equivalence of circuits obtained in this way could be easily checked via processing arbitrary inputs through both circuits. Two fundamental examples are



and



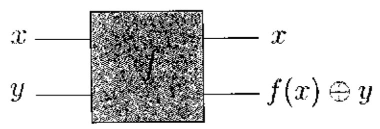
where the first one is simply a *controlled-Not* (unitary matrix C above) and the second is the *Toffoli* gate.

Deutsch-Jozsa's Algorithm

We will now examine a quantum algorithm which is clearly superior to any possible classical algorithm for the same job: Let us say that we have a "black box program" which takes as input an n-bit string, and computes and outputs a boolean (0 or 1) function f of this input. We can give any input that we like to this program, and examine the output, but we are not allowed to look inside at the code of the program. Such black box subroutines are called *oracles* in the theory of computation. The only thing that we know about the function f (in this specific example) is that it is either a constant function (i.e. it gives the same output for every input) or a balanced function (i.e. it gives output 0 for half of the possible inputs, and 1 for the other half). Our task is to determine whether f is constant or balanced.

Obviously, the worst-case complexity of the best classical algorithm for this job is $2^{O(n)}$. You need to run the black box at least $2^{n-1}+1$ times with different inputs in the worst case. We will now see a quantum algorithm that can do this job with a single run of the black box (The case $n=1$ is called the *Deutsch algorithm*).

Our algorithm must be given a quantum version of the black box. As we mentioned above, for any job that has a classical algorithm that inputs n bits and outputs m bits, one can write a quantum algorithm that inputs and outputs $n+m$ qubits. So let us say that we have a quantum program ***B*** which operates on $n+1$ qubits, such that the input $|x\rangle \otimes |d\rangle$ is transformed by ***B*** to the output $|x\rangle \otimes |d \oplus f(x)\rangle$, where $\oplus$ is the exclusive-or operator.



Here is an algorithm that operates on $n+1$ qubits to solve our problem:

1. Initialize the register so that the first n qubits are $|0\rangle$, and the last one is $|1\rangle$.
2. Apply the H gate to each qubit.
3. Apply ***B*** to the register.
4. Apply the H gate to the first n qubits.
5. Read (measure) the number z written in the first n qubits.
6. If $z=0^n$, f is constant, otherwise, f is balanced.

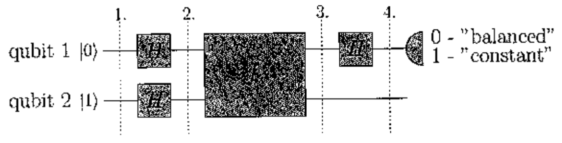


Figure 2: Deutsch's circuit.

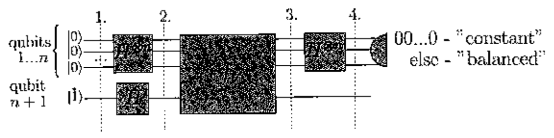


Figure 3: Deutsch-Jozsa's circuit.

Let us see why the algorithm works correctly: At the end of stage 2, the state of the register is

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

At the end of stage 3, we have, for each x in the summation, such a term of the state in the register:

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

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

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

So the overall state after stage 3 is:

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

After stage 4, we end up with:

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

Now, for any number z, the probability that we see it at stage 5 is the square of its amplitude, that is,

$$\left|\frac{1}{2^n}\sum_{x=0}^{2^n-1}(-1)^{x\cdot z+f(x)}\right|^2.$$

So the probability of seeing 0^n is

$$\left|\frac{1}{2^n}\sum_{x=0}^{2^n-1}(-1)^{f(x)}\right|^2=\begin{cases}1 & \text{if f is constant}\\ 0 & \text{if f is balanced.}\end{cases}$$

This algorithm, called the *Deutsch-Jozsa algorithm*, was one of the first that demonstrated the advantage of quantum algorithms over classical ones.

Simon's Algorithm

Let us now examine Simon's Algorithm. Unlike the Deutsch-Jozsa algorithm, Simon's algorithm has a nonzero error probability, that is, it is not 100% certain to give the correct answer. However, the probability that it gives the correct answer is bigger than ½ if some mild conditions are satisfied.

Simon's Problem: Given a black box U_f which computes a function $f:\{0,1\}^n\to\{0,1\}^m \quad (m\geq n)$ that is known either to be one to one or to satisfy the equation

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

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:

$$U_f(|x\rangle\otimes|d\rangle)=|x\rangle\otimes d\oplus f(x)\rangle))$$

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

Step 0: Initialize two quantum registers of n and m qubits in the states $\overset{n-1}{\underset{0}{\otimes}}|0\rangle$ *and* $\overset{m-1}{\underset{0}{\otimes}}|0\rangle$.

Step 1: Apply n-bit Hadamard gate H_n to the first register of n bits. The overall state will be as shown in the following equation. Note that this step puts the first register into an equal superposition of the 2^n basis states.

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

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

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

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

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

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, $2^n2^n=2^{2n}$ states of the form $|j\rangle\otimes|f(k)\rangle$ 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 probabilities 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

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

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.

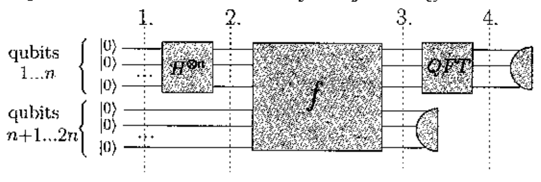


Figure 4: Simon's circuit.

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. Consider any (i-1)-member prefix of our sequence of observations, namely $t_1, t_2, \ldots, t_{i-1}$, 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 $\oplus$. 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

$$\left(1-\frac{1}{2^{n-1}}\right)\left(1-\frac{1}{2^{n-2}}\right)L\left(1-\frac{1}{2}\right)..$$

which can be shown to be greater than ¼.

So, if the equations we obtained really are linearly independent, this system of n-1 linear equations in n unknowns can be solved for the 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 in solving 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.

Clearly, 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 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 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 which 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 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

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

In order to be able to say that "this algorithm solves the problem with probability at least p", for a constant p. We clearly have to set m to a value around at least $2^{n/2}$, which is exponential in terms of n. Note that p shouldn't decrease when n grows, since 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.

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 finding algorithm turns out to be useful for developing algorithms for many other problems.

Grover's Algorithm

We will now examine Grover's algorithm for function inversion, which can be used 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). Assume that we are given a quantum oracle $\boldsymbol{G}$ 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 example, the oracle operates on n+1 qubits, 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 qubit register, where n>2:

1. Initialize the register so that the first n qubits are $|0\rangle$, and the last one is $|1\rangle$.
2. Apply the H gate to each qubit.

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*, which will be described below, to the first n qubits.
4. Read (measure) the number written in the first n qubits, and claim that this is the value which makes *f* equal 1.

The program *V* is defined as the 2^n by 2^n matrix which 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.

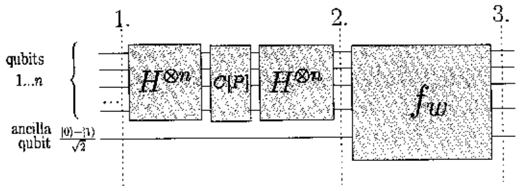


Figure 5: Grover's circuit

Shor's Factorization Algorithm

The most famous potential application of quantum computing is embodied in Shor's algorithm, which can find a factor of a given binary integer in a polynomial number of steps, which is exponentially faster than the best known classical algorithm. The pseudo code of the algorithm is as follows.

On input N: (N is known to be composite number with n bits, you can check for primality in polynomial time classically anyway)
If N is even, print 2, end.

For every $2 \le b \le \log N$
Perform binary search in the interval [2,N] for an a that satisfies $a^b = N$, if you find one, print a, end.

NOTPERFPOWER:
Randomly pick an integer x in {2...N–1}. (This can be done using a quantum computer.)
If gcd(x,N)>1 then print gcd(x,N), end. (Fast calculation of the gcd is easy, using Euclid's famous algorithm.)

ORDFIND:
Let t be 2n+1.
Build (don't run right now) a quantum program called E, which operates on two registers (of t and n qubits, respectively,) and which realizes the transformation $|j\rangle|k\rangle \rightarrow |j\rangle|x^j k \bmod N\rangle$.
Do the following 5log(N) times:

QINIT:
Initialize the first quantum register of t qubits to $|0\rangle$, and the second quantum register of n qubits to $|1\rangle$.
Apply the H gate to each qubit in the first register.
Apply the program E to the combination of the first and second registers.
Apply a program which realizes the inverse of the transformation $QFT|j\rangle = \frac{1}{\sqrt{N}}\sum_{k=0}^{N-1} e^{2\pi\cdot i\cdot j\cdot k/N}|k\rangle$ to the first register.

Measure the first register to obtain a number m.

Apply the classical "continued fractions" algorithm, to find irreducible fractions of the form num/den, which approximate the number $m/2^t$: This algorithm works in a loop, and it prepares a new fraction num/den which is a closer approximation to $m/2^t$ at the end of each iteration. Here r is initialized with a large value, say $2N^2+1$. At the end of each iteration, we take the new den value and do the following:
If den > N then goto QINIT
If $x^{den} = 1 \pmod N$
then BEGIN
if r has not been assigned a smaller value than this den earlier, then let r be den
Go to QINIT
END
If none of these conditions are satisfied, we continue with the next iteration of the continued fractions loop, which will prepare a new and better num/den with a greater value for den.
If no new value has been assigned to r, print "failed", end.
LASTST: If r is odd or $x^{r/2} = -1 \pmod N$, print "failed", end.
Check if $\gcd(x^{r/2}+1, N)$ or $\gcd(x^{r/2}-1, N)$ is a nontrivial factor of N, if so, print that nontrivial factor, if not, print "failed", end.

Note that only the "middle part" (stages QINIT to the measurement) of this algorithm actually involves qubits.

Why does this work?

We handle the case where *N* is a perfect power (i.e. the power of a single integer) separately. Note that if $a^b = N$ for integers *a* and *b* and *N*>1, then *b* can be at most log *N*. This is such a small number that we can try all possible values for *b* and remain within a polynomial bound. Finding whether an *a* exists for a particular *b* can be done by binary search over the space of all possible *a*'s (another very efficient algorithm) where we just raise the candidates to the b^{th} power to see if the result equals *N*. Since the *b*'s are so small, this exponentiation can be done in polynomial time.

The job of the part of the program starting with the stage ORDFIND is to find what mathematicians call "the order of x modulo N," i.e. the least positive integer r such that x^r=1(mod N) for an x which is chosen randomly from the positive integers less than N which are co-prime to N, that is, gcd(x,N)=1. The program reaches the stage LASTST only when that r has been found for x and N. Now, if that r survives the additional checks specified in that line, at least one of gcd($x^{r/2}$ + 1, N) , gcd($x^{r/2}$ − 1, N) is guaranteed to be a nontrivial factor of N.

Because if r is the order of x modulo N and r is even, then $x^{r/2}$(mod N) (let's call it y) is definitely an integer which is not equal to 1(mod N), and y^2=1(mod N). This can be rewritten as y^2–1=0(mod N), and since we clearly have $y\neq -1$ = N–1 (mod N), also as (y–1)(y+1)=0 (mod N), such that both y–1 and y+1 are nonzero. This means that N has a factor in common with y–1 or y+1. Furthermore, that factor cannot be N itself, since the previously mentioned constraints mean that both of y+1 and y+1 are positive integers less than N. So when we compute gcd($x^{r/2}$ + 1, N) and gcd($x^{r/2}$–1,N), at least one of them will be a nontrivial factor of N.

Now, all we need to show is that the ORDFIND stage of the program really finds the required order, and that all this happens in polynomial time with high probability. To do this, we introduce a quantum algorithm for solving a more general problem: That of finding the phase of the eigenvalue of a particular quantum program. Here are the definitions for the terminology in the previous sentence: $|v\rangle$ is an eigenvector and the complex number a is an eigenvalue of a quantum program U if they satisfy

$U|v\rangle = a|v\rangle$.

In the phase estimation problem, we assume that we are given a quantum program E which performs the transformation $E(|j\rangle|u\rangle) \rightarrow |j\rangle U^j|u\rangle$ for integer j for the program U in which we are interested. We are also given $|v\rangle$, which is guaranteed to be an eigenvector of U. With this much information, it is certain that the eigenvalue corresponding to this eigenvector is of the form $a = e^{i2\pi\varphi}$, and our task is to find the real number φ, called the phase, which is between 0 and 1.

The phase estimation algorithm is carried out as follows: We initialize the first register to $|0\rangle$. We initialize the second register to $|v\rangle$. We apply the H gate to all qubits of the first register. We apply the program E to the register pair. Finally, we apply the inverse quantum Fourier transform to the first register, and then measure the first register.

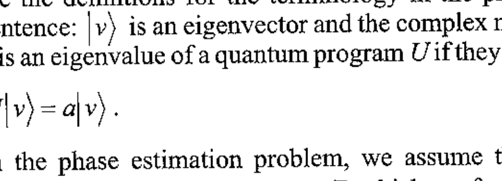


Figure 6: *QFT* on Z_8. elements of Z_8 is represented in binary notation $x = x_2x_1x_0$, $y = y_2y_1y_0$.

By dividing the integer we read by the number 2^t, where t is the number of qubits in the first register, we obtain a pretty good approximation to φ with pretty high probability of correctness.

The way we explained it above, one has to know the number of solutions (M) to the problem before one can use Grover's algorithm to find a solution. However, a clever combination of the iterate used in that algorithm and the phase estimation technique we saw in the discussion of Shor's algorithm can be used to estimate M using only $\Theta(\sqrt{N})$ oracle calls, where N is as defined in the discussion of Grover's algorithm. This means that a search-based quantum algorithm for solving NP-complete problems will run quadratically faster than a search-based classical algorithm.

Computability

Can quantum computers compute all functions that can be computed by classical computers? The answer is yes, and is based on the fact that the three-qubit *Toffoli* gate can be used to implement any classical computation, as proven in, as usual, Nielsen & Chuang (2000). In fact, just the Toffoli and Hadamard gates form another quantum-universal set of gates.

Can quantum computers compute functions that classical ones (i.e. Turing machines) cannot? If so, wouldn't that be the end of the theory of computation as we learned it?

Recall that we have defined a quantum algorithm as a unitary matrix of complex numbers. We do not know of any law of physics that forbids any such matrix from having an actual physical implementation, so it may be the case that actual physical entities which are embodiments of quantum programs which contain arbitrary transcendental numbers of infinite precision in their matrices exist. Note that this violates one of the assumptions that we made about Turing machines; namely, that they must be completely describable by a finite string. We now show that, for *every* language (even for the classically undecidable ones like A_{TM}) there exists a quantum algorithm which can decide that language if we allow a small probability of error in the answers. This is impossible for classical computers, even if you let them use random numbers and allow the small error probability.

We know that the set of all strings on a given nonempty alphabet can be put in one-to-one correspondence with the set of positive integers, and it is easy to write a program that computes the integer corresponding to a given string. We use this idea in the first stage of the following program which decides language L with bounded error.

On input w:

1. Compute the integer corresponding to w in the lexicographic ordering and assign it to variable j.

2. Let i be $j-1$.
3. Set a single qubit to $|0\rangle$.
4. Apply the ***R*** gate, which will be described below, 8^i times to this qubit.
5. Apply the gate $\begin{pmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix}$ to this qubit.
6. Observe this qubit. If you see $|1\rangle$, *accept*; otherwise, *reject*.

The ***R*** gate is the matrix $\begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$, where the number $\theta = 2\pi\left(\sum_{x=1}^{\infty} \frac{H(x)}{8^x}\right)$, such that the function H from the positive integers to the set {-1, 1} is defined to be 1 if the string x is in L and -1, otherwise.

The secret of the success of this program is, of course, its ability to encode the membership function of the entire language in the digits of the real number θ. Clearly, even if we had a universal quantum computer on which we could implement any quantum program that we can write, we would not be able to use it to solve one of the classically undecidable problems, because we just do not know which θ to use!

To avoid this confusion, most researchers limit their discussions to quantum programs which contain only computable numbers (i.e. numbers that have classical Turing machines which can print their digits to any desired degree of precision) in their matrices. With this restriction, the quantum programs have finite descriptions in the TM language, and can therefore be simulated (within any given nonzero upper bound for deviations) by classical TMs, meaning that they cannot solve any classically undecidable problem.

Complexity Classes

The class of languages that can be decided by quantum computers (with the above-mentioned restriction) with bounded error in polynomial time is called BQP. It is known that P⊆BQP⊆PSPACE. It is not known whether any of these inclusions is strict.

Recent Developments

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

Quantum Walks

One of the biggest breakthroughs in classical algorithm design was the introduction of randomness and the notion of a probabilistic algorithm. Many problems have good algorithms that use a random walk as a subroutine. To give just one example, the currently best algorithm to solve 3SAT is based on a random walk.

With this motivation in mind, *quantum* analogues of *random walks* have been introduced. There exist two different models of a quantum walk, the continuous-time model and the discrete time model. The continuous model gives a unitary transformation directly on the space on which the walk takes place. The discrete model introduces an extra coin register and defines a two-step procedure consisting of a "quantum coin flip" followed by a coin-controlled walk step.

The quantities important for algorithm design with random walks are their mixing time (the time it takes to be close to uniformly distributed over the domain) and the hitting time (the expected time it takes to hit a certain point). These quantities have been analyzed for several graphs in both the continuous and the discrete model.

It turns out that a quantum walk can speed up the mixing time up to quadratic with respect to its classical counterpart; so the classical and quantum performance are polynomial related. The hitting behavior of a quantum walk, however, can be very different from classical. It has been shown that there are graphs and two vertices in them such that the classical hitting time from one vertex to the other is polynomial in the number of vertices of the graph, whereas the quantum walk is exponentially faster. It is open whether quantum hitting times can be used to speed up classical algorithms for relevant problems.

Adiabatic Quantum Algorithms

Another recent alternative for algorithm design has been the introduction of *adiabatic quantum algorithms* by Farhi et al. The idea is the following: many optimization and constraint satisfaction problems can be encoded into a sum of local Hamiltonians such that each term represents a local constraint. The ground state of H violates the smallest number of such constraints and represents the desired optimal solution. In order to obtain this state, another Hamiltonian is chosen so that the corresponding ground state is easy to prepare.

An adiabatic algorithm starts in the initial state and applies the easy Hamiltonian. The Hamiltonian is then slowly changed to the original one, usually in a linear fashion over time, such that the Hamiltonian $H(t)$ at time t is a convex combination of the new and old Hamiltonians. If this is done slowly enough, the adiabatic theorem guaranties that the state at time t will be the ground state of $H(t)$, leading to the solution, the ground state of H, at the running time T. The instantaneous ground state of the system is *tracked.*

Conclusion

The search for new quantum algorithms continues. Although an adiabatic algorithm can be simulated efficiently with a quantum circuit (one needs to implement a time-dependent unitary that is given by a set of local Hamiltonians, each one acting only on a few qubits) and vice-versa, and therefore these two models of computation are essentially equivalent, the advantage of the adiabatic model is that it deals with gaps of Hermitian matrices, an area that has been widely studied both by solid state physicists and probabilists. This has caused some hopes for this new toolbox to yield new algorithms.

Acknowledgement

The treatment of algorithms in this article is based on lecture notes of Prof. *Ahmet Celal Cem Say*, from the Department of Computer Science in *Boğaziçi University*, Turkey. The recent development is taken from the lecture notes of Prof. *Julia Kempe*, in the *Universite de Paris* summer school in 2006.

References

[1] Adleman, L., DeMarrais, J., Huang, M. 1997. Quantum Computability, *SIAM Journal on Computing,* **26**, 1524-1540.

[2] Ambainis, A. and Watrous, J. 2002. Two-way finite automata with quantum and classical state, *Theoretical Computer Science,* **287**, 299-311.

3] Benenti, G., Casati, G., Strini, G. 2004. *Principles of Quantum Computation and Information.* Singapore: World Scientific.

[4] Bernstein, D. J. 1998. Detecting perfect powers in essentially linear time, *Math. Comp.,* **67,** 1253-1283.

[5] Childs, A. M., Cleve, R., Deotto, E., Farhi, E., Gutmann, S., Spielman, D. A. 2003. Exponential Algorithmic Speedup by a Quantum Walk. *STOC'03*, 59-68.

[6] Farhi E., Goldstone, J., Gutmann, S., Sipser, M. 2000. Quantum Computation by Adiabatic Evolution, Quant-Ph/0001106.

[7] Gruska, J. 1999. *Quantum Computing.* New York: McGraw Hill.

[8] Kempe, J. 2006. *Quantum Algorithms.* Summer School on Theory and Technology in Quantum Information, Communication, Computation and Cryptography, Universite de Paris-Sud.

[9] Kitaev, A. Yu., Shen, A. H., Vyalyi, M. N. 2002. *Classical and Quantum Computation.* American Mathematical Society.

[10] Moore, C., Crutchfield, J. P. 2000. Quantum Automata and Quantum Grammars, *Theoretical Computer Science,* **237**, 275-306.

[11] Nielsen, M. A., Chuang, I. L. 2000. *Quantum Computation and Quantum Information.* Cambridge, New York: Cambridge University Press.

[12] Rudolph, T. and Grover, L. A 2 Rebit Gate Universal for Quantum Computing. quant-ph/0210187.