Introduction to Quantum Computing

Javier Orduz
CSI 5V93

August 11, 2021

Contents

Part I

Quantum Circuits & Quantum Algorithms (Basic Qiskit)

In this chapter, we talk about ...

Chapter 1

Objectives

The chapter's objective is

General objective

Represent algorithms with quantum logical gates with Qiskit framework (or anyother).

Chapter 2

Activities, materials and more

In this chapter you will use:

- Computer
- Mobile phone
- tablet
- Github, Gitlab, Colab,
- IBM lab: IBM lab-Cognitive
- Strangeworks Strangeworks
- SILQ: SILW
- Pennylane
- Dwave
- Q#
- others

In activities section, we will use:

- Kahoot
- Padlet
- GForms
- etc.

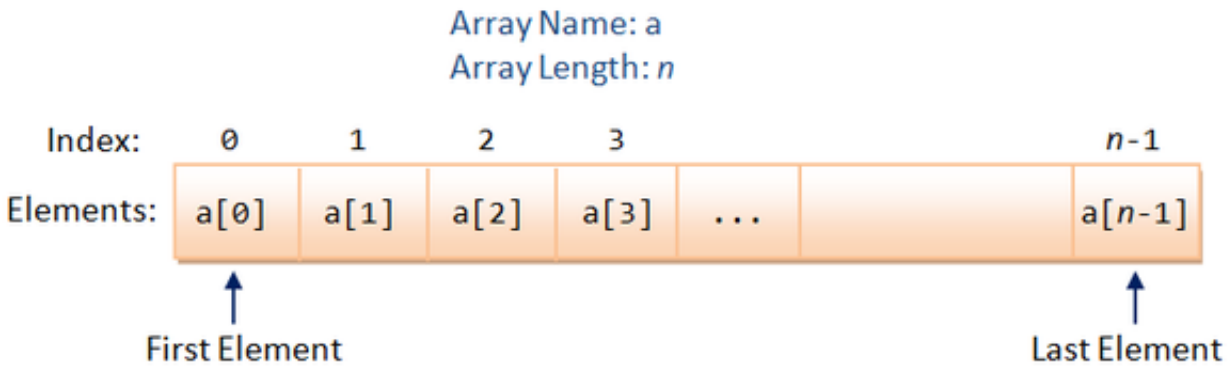
Chapter 3

Algorithms

If you want to see one timeline of QC go to this link TimeLine (prezi)

3.1 Black Box: Oracle Function

Suppose we wish to search through a search space of N elements.



Rather than search the elements directly, we concentrate on the **index** to those elements, which is just a number in the range 0 to $N - 1$.
For convenience we assume

- $N = 2^n$ so the index can be stored in n bits,
- and that the search problem has exactly M solutions, with $1 \leq M \leq N$.

We can propose a function $f(x)$ to represent some particular cases, this function can take input (integer, x) in the range $\{0, N - 1\}$.

Definition

$$f(x) = \begin{cases} 1 & \text{solution} \\ 0 & \text{No solution} \end{cases} \tag{3.1}$$

to the search problem.

Oracle function:

Oracle functions

Since it is assumed that one does not have access to the inner workings of the computation of this function, only to the result of its application.

Suppose we are supplied with a quantum oracle − a black box whose internal workings, but which it is not important at this stage − with the ability to recognize solutions to the search problem. This recognition is signalled by making use of an oracle qbit.
More precisely, the oracle is a unitary operator, $\mathcal{O}$, defined by its action on the computational basis:

$$|x\rangle|q\rangle \xrightarrow{\mathcal{O}} |x\rangle|q \oplus f(x)\rangle \tag{3.2}$$

where

1. $|x\rangle$ is the index register,
2. $\oplus$ denotes addition modulo 2,
3. and the oracle qubit $|q\rangle$ is a single qbit which is flipped if $f(x) = 1$, and is unchanged otherwise, see eq. (3.1).

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 oracle qbit has been flipped to $|1\rangle$.
Oracle function is important in different algorithms.

Example 1 *Oracle: Deutsch-Jozsa algorithm*
Consider next steps

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

2. *If x is not a solution to the search problem, applying the oracle to the state*

$$\frac{|x\rangle\,(|0\rangle - |1\rangle)}{\sqrt{2}}$$

does not change the state.

3. *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*

$$-\frac{|x\rangle\left(|0\rangle-|1\rangle\right)}{\sqrt{2}}.$$

4. *The action of the oracle is thus:*

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

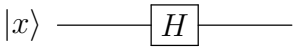
With an oracle we want determine some property of the oracle using the minimal number of queries. In summary, an oracle is a physical device that we cannot look inside, we use to pass queries and get answers. Besides, on a quantum computer, an oracle must be reversible.

3.2 Hadamard n

Recall

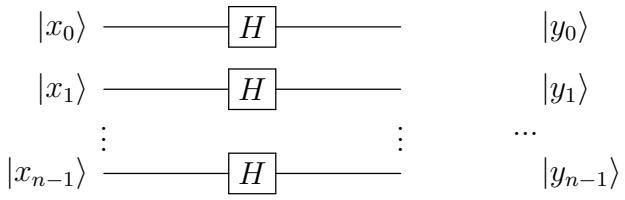
$$\begin{aligned} H|0\rangle &= |+\rangle \\ H|1\rangle &= |-\rangle \end{aligned} \tag{3.3}$$

Then we have



$$= \tfrac{1}{2^{1/2}} \sum_{k\in\{0,1\}} (-1)^{k\odot x}\,|k\rangle$$

Then in general,



$$|y\rangle = \tfrac{1}{2^{n/2}} \sum_{k\in\{0,1\}^n} (-1)^{k\odot x}\,|k\rangle$$

where

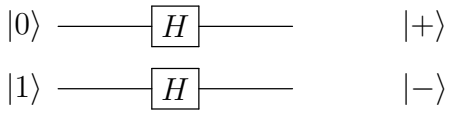
$$|y\rangle = H^{\otimes n}\,|x\rangle = \frac{1}{2^{n/2}} \sum_{k\in\{0,1\}^n} (-1)^{k\odot x}\,|k\rangle \tag{3.4}$$

and

$$k\odot x = (k_2)_{n-1}(x_2)_{n-1} + \ldots + (k_2)_1(x_2)_1 + (k_2)_0(x_2)_0 \tag{3.5}$$

with k_2 is the k represented in binary.
Eq. (3.4) is the superposition of all 2^n bit strings states.

Example 2 $|x\rangle\,|01\rangle$ *Then*



Considering eq. (3.4),

$$\begin{aligned}
|y\rangle = H^{\otimes 2}\,|x\rangle &= \frac{1}{2^{n/2}}\sum_{k\in\{0,1\}^2}(-1)^{k\odot x}\,|k\rangle \\
&= \frac{1}{2}\Big((-1)^{k\odot x}\,|00\rangle + (-1)^{k\odot x}\,|01\rangle + (-1)^{k\odot x}\,|10\rangle + (-1)^{k\odot x}\,|11\rangle\Big) \\
&= \frac{1}{2}\Big((-1)^{(0,0)\odot(0,1)}\,|00\rangle + (-1)^{(0,1)\odot(0,1)}\,|01\rangle + (-1)^{(1,0)\odot(0,1)}\,|10\rangle + (-1)^{(1,1)\odot(0,1)}\,|11\rangle\Big) \\
&= \frac{1}{2}\Big(\,|00\rangle - |01\rangle + |10\rangle - |11\rangle\Big)
\end{aligned}$$

$$\frac{1}{2^{1/2}}\sum_{k\in\{0,1\}^2}(-1)^{k\odot x}\,|k\rangle \tag{3.6}$$

In general,

$$H^{\otimes n} = \frac{1}{\sqrt{2^n}}\sum_{x,y}(-1)^{x\cdot y}\,|x\rangle\,\langle y| \tag{3.7}$$

3.3 Deutsch's algorithms

Deutsch's algorithm [3]

Deutsch showed that his quantum algorithm has better query complexity than any possible classical algorithm: it can solve the problem with fewer calls to the black box than is possible classically.

Given a Boolean function

$$f : \mathbb{Z}_2 \to \mathbb{Z}_2,$$

determine whether f is constant.

Problem

Input: A black box for computing an unknown function function $f : \{0,1\} \to \{0,1\}$.

Problem: Determine the value of $f(0) \oplus f(1)$ by making querios of f.

Deutsch's algorithm provides the first example of a truly quantum algorithm, one for which there is no classical analog, and it was the first few quantum algorithms solve black box problems [3].
The earliest quantum algorithms solve black box, or oracle, problems.

Black Box problem

A classical black box outputs $f(x)$ upon input of x. A quantum black box behaves like U_f, outputting $\sum_x \alpha_x |x, f(x) \oplus y\rangle$ upon input of $\sum_x \alpha_x |x\rangle|y\rangle$. Black boxes are theoretical constructs; they may or may not have an efficient implementation. For this reason, they are of ten called oracles. The black box terminology emphasizes that only the output of a black box can be used to solve the problem, not anything about its implementation or any of the intermediate values computed along the way; we cannot see inside it. The most common type of complexity discussed with respect to black box problems is query complexity: how many calls to the oracle are required to solve the problem.
Black box algorithms of low query complexity, algorithms that solve a black box problem with few calls to the oracle, are only of practical use if the black box has an efficient implementation. The black box approach is very useful, however, in establishing lower bounds on the circuit complexity of a problem. If the query complexity is $\Omega(N)$ -in other words, at least $\Omega(N)$ calls to the oracle are required - then the circuit complexity must be at least $\Omega(N)$.
Black boxes have been used to establish lower bounds on the circuit complexity for quantum algorithms, but their first use in quantum computation was to show that the quantum query complexity of certain black box problems was strictly less than the classical query complexity: the number of calls to a quantum oracle needed to solve certain problems is strictly less than the required number of calls to a classical oracle to solve the same problem.
The first few quantum algorithms solve black box problems: Deutsch's problem (section 7.3.1), the Deutsch-Jozsa problem (section 7.5.1), the Bernstein-Vazirani problem (section 7.5.2), and Simon's problem (section 7.5.3). The most famous query complexity result is Grover's: that it takes only $O(\sqrt{N})$ calls to a quantum black box to solve an unstructured search problem over N elements, where as the classical query complexity of unstructured search is $\Omega(N)$. Grover's algorithm, and the extent to which its superior query complexity provide practical benefit, are discussed in chapter 9 [3, pag. 131].

The problem Deutsch's algorithm solves is a black box problem. Deutsch showed that his quantum algorithm has better query complexity than any possible classical algorithm. it can solve the problem with fewer calls to the black box than is possible classically.

Deutsch's quantum algorithm , described in this section, requires only a single call to a black box for U_f to solve the problem.

Any classical algorithm requires two calls to a classical black box for C_f, one for each input value.

The key to Deutsch's algorithm is the nonclassical ability to place the second qubit of the input to the black box in a superposition.

Since, whether we apply U_f on a single bit function f takes two qbits of input and produces two qbits of output. On input $|x\rangle\,|y\rangle\,, U_f$ produces

$$|x\rangle\,|f(x)\oplus y\rangle\,,$$

so when

$$|y\rangle = |0\rangle\,,$$

the result of applying U_f is $|x\rangle|f(x)\rangle$.
The algorithm applies U_f to

- the two-qubit state $|+\rangle\,|-\rangle$, where the first qubit is a superposition of the two values in the domain of f,
- and the third qubit is in the superposition $|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$. We obtain

$$\begin{aligned}
U_f(|+\rangle|-\rangle) &= U_f\left(\frac{1}{2}\Big(\,|0\rangle + |1\rangle\,\Big)\Big(\,|0\rangle - |1\rangle\,\Big)\right)\\
&= \frac{1}{2}U_f\Big(\,|0\rangle\,|0\rangle - |1\rangle\,|0\rangle + |1\rangle\,|0\rangle - |1\rangle\,|1\rangle\,\Big)\\
&= \frac{1}{2}\Big(\,U_f\,|0\rangle\,|0\rangle - U_f\,|0\rangle\,|1\rangle + U_f\,|1\rangle\,|0\rangle - U_f\,|1\rangle\,|1\rangle\,\Big)\\
&= \frac{1}{2}\Big(\,|0\rangle\,|0\oplus f(0)\rangle - |0\rangle\,|0\oplus f(0)\rangle + |1\rangle\,|0\oplus f(1)\rangle - |1\rangle\,|1\oplus f(1)\rangle\,\Big)\\
&= \frac{1}{2}\Big(\,|0\rangle\,\Big(\,|0\oplus f(0)\rangle - |1\oplus f(0)\rangle\,\Big) + |1\rangle\,\Big(\,|0\oplus f(1)\rangle - |1\oplus f(1)\rangle\,\Big)\Big) \qquad (3.8)
\end{aligned}$$

In other words,

$$U_f(|+\rangle|-\rangle) = \frac{1}{2}\sum_{x=0}^{1}|x\rangle(|0\oplus f(x)\rangle - |1\oplus f(x)\rangle) \qquad (3.9)$$

1. When $f(x) = 0$

$$\frac{1}{\sqrt{2}}(|0\oplus f(x)\rangle - |1\oplus f(x)\rangle)$$

becomes

$$\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) = |-\rangle.$$

2. When $f(x) = 1$,
$$\frac{1}{\sqrt{2}}(|0 \oplus f(x)\rangle - |1 \oplus f(x)\rangle)$$
becomes
$$\frac{1}{\sqrt{2}}(|1\rangle - |0\rangle) = -|-\rangle.$$

Therefore

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

1. For f constant, $(-1)^{f(x)}$ is just a physically meaningless global phase, so the state is simply $|+\rangle|-\rangle$.

2. For f not constant, the term $(-1)^{f(x)}$ negates exactly one of the terms in the superposition so, up to a global phase, the state is $|-\rangle|-\rangle$.

3. If we apply the Hadamard transformation H to the first qbit and then measure it, with certainty we obtain $|0\rangle$ in the first case and $|1\rangle$ in the second case. Thus with a single call to U_f we can determine, with certainty, whether f is constant or not.

This example shows that a quantum algorithm performs well, compares to any classical algorithm!
The algorithm for Deutsch's problem shows that even inherently quantum processes do not have to be probabilistic.

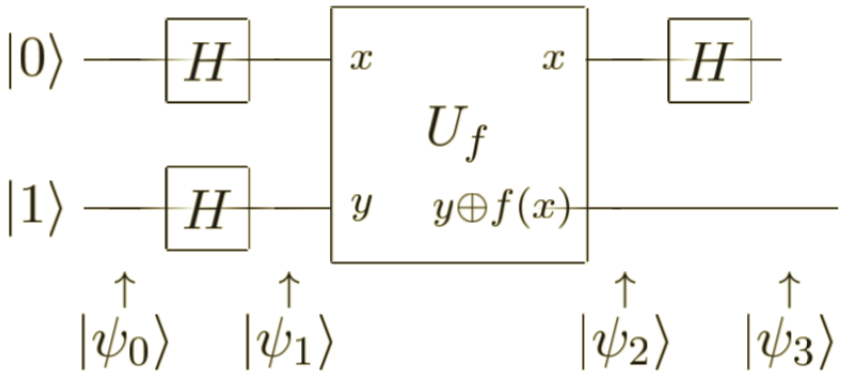


Figure 3.1: Quantum circuit implementing Deutsch's algorithm

where

$$\begin{aligned} |\psi_0\rangle &= |01\rangle \\ |\psi_1\rangle &= \left[\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right]\left[\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right] \end{aligned}$$

A little thought shows that if we apply U_f to the state

$$\frac{|x\rangle(|0\rangle - |1\rangle)}{\sqrt{2}}$$

then we obtain the state

$$\frac{(-1)^{f(x)}|x\rangle(|0\rangle - |1\rangle)}{\sqrt{2}}.$$

Applying U_f to $|\psi_1\rangle$ therefore leaves us with one of two possibilities:

$$|\psi_2\rangle = \begin{cases} \pm\left[\frac{|0\rangle+|1\rangle}{\sqrt{2}}\right]\left[\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right] & \text{if } f(0) = f(1) \\ \pm\left[\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right]\left[\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right] & \text{if } f(0) \neq f(1) \end{cases}$$

$$|\psi_3\rangle = \begin{cases} \pm|0\rangle\left[\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right] & \text{if } f(0) = f(1) \\ \pm|1\rangle\left[\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right] & \text{if } f(0 \neq f(1) \end{cases}$$

But $f(0) \oplus f(1) = 0$ if $f(0) = f(1)$ and 1 otherwise. We can write,

$$|\psi_3\rangle = \pm|f(0) \oplus f(1)\rangle\left[\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right] \tag{3.10}$$

It means if we measure the first qbit we may determine $f(0) \oplus f(1)$.
This algorithm uses

- Quantum parallelism
- Interference
- Superposition

We prepare two qbits: $|+\rangle$ and $|-\rangle$. And apply $H-$gate on

3.4 The Deutsch–Jozsa algorithm

Problem

Input: A black-box for computing an unknown function function $f : \{0,1\}^n \to \{0,1\}$.

Promise: f is either a constant or a balanced function.

Problem: Determine whether f is constant or balanced by making queries to f.

The Deutsch-Jozsa algorithm is a generalization of Deutsch's algorithm. Again, this algorithm allows us to determine whether a function $f(x)$ is constant or balanced[1], but this time the function has multiple input values. If $f(x)$ is constant, then the output is the same for all input values x. If the function is balanced, then $f(x) = 0$ for half of the inputs and $f(x) = 1$ for the other half of the inputs, and vice versa. We start with an initial state that includes n qubits in the state $|0\rangle$ and a singlequbit in the state $|1\rangle$. Hadamard gates are applied to all qubits. The circuit is illustrated in Figure 3.2.

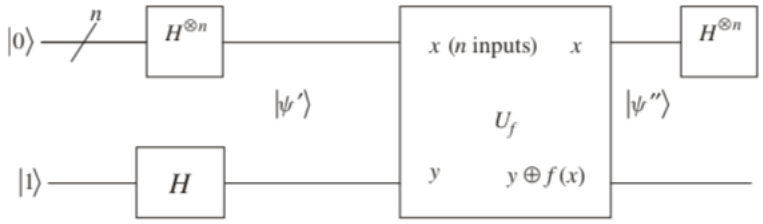


Figure 3.2: The Deutsch-Jozsa algorithm generizes Deutsch's algorithm to handle a function with n input values and determine whether or not it is constant or balanced.

We start off by calculating

$$|\psi'\rangle = \left(H^{\otimes n}\right)\left(|0\rangle^{\otimes n}\right) \otimes (H|1\rangle).$$

From

$$H^{\otimes n}\left(|0\rangle^{\otimes n}\right) = \frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} |x\rangle$$

we see that this is

$$|\psi'\rangle = \frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} |x\rangle \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right)$$

Next we apply $U_f\,|x, y\rangle = |x, y \oplus f(x)\rangle$, to evaluate the function. The first n qubits are the values of x and the last qubit plays the role of y as shown in the figure. The output state of the U_f gate is

$$|\psi''\rangle = \frac{1}{\sqrt{2^n}} \sum_x (-1)^{f(x)} |x\rangle \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right).$$

Applying a Hadamard gate to an n qubit state $|x\rangle$ gives

$$H^{\otimes n}|x\rangle = \frac{1}{\sqrt{2^n}} \sum_y (-1)^{x\cdot y} |y\rangle$$

[1]The identity and bit flip functions are called balanced because the outputs are opposite for half the inputs. So a function on a single bit can be constant or balanced. Whether a function on a single bit is constant or balanced is a global property. What we're going to see in the following development is that Deutsch's algorithm will let us put together a state that has all of the output values of the function associated with each input value in a superposition state. Then we will use quantum interference to find out if the given function is constant or balanced.

So the final output state is

$$|\psi_{\text{out}}\rangle = \frac{1}{2^n}\sum_y\sum_x(-1)^{x\cdot y+f(x)}|y\rangle\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right)$$

Now we measure the n inputs. It might not be immediately obvious looking at $|\psi_{\text{out}}\rangle$, but there are two possible measurement results on $|y\rangle$ (which is the state of n inputs at this stage) that are of interest. The possible results are as follows: Measurement of the first n input qubits in $|\psi_{\text{out}}\rangle$ returns all 0's. In this case $f(x)$ is constant. Otherwise, if at least one of the qubits in $|y\rangle$ is found to be a 1 on measurement, $f(x)$ is balanced[2].
Other way,

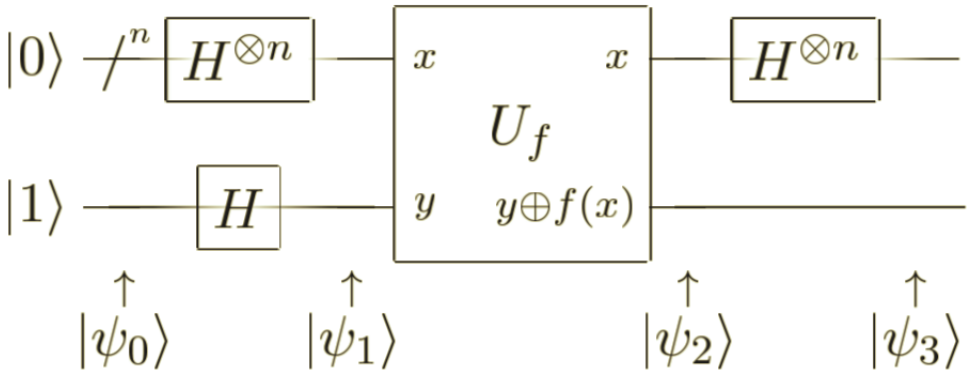


Figure 3.3: Quantum circuit implementing Deutsch-Jozsa algorithm

3.5 Grover's algorithm: Quantum search algorithm

The search Problem

Input: A black box U_f for computing an unknown function $f : \{0,1\}^n \to \{0,1\}$.

Problem: Find an input $x \in \{0,1\}^n$ such that $f(x) = 1$ [1, pag. 152].

This algorithm enables this search method to be speed up to $\mathcal{O}(\sqrt{N})$ operations.
With this algorithm ´´searching an unsorted database" with $N = 2^n$ elements in $\mathcal{O}(\sqrt{N})$ time. Classical algorithm needs on average $N/2 = \mathcal{O}(\sqrt{N})$ time. The goal is find w, given an oracle U_f with

$$f : \{0,1\}^n \to \{0,1\}$$

$$f(x) = \begin{cases} 1 & \text{if } x = w \\ 0 & \text{else if} \end{cases} \tag{3.11}$$

and

$$f_0(x) = \begin{cases} 0 & \text{if } x = 0000 \\ 1 & \text{else} \end{cases} \tag{3.12}$$

where the phase oracle is

$$U_f |x\rangle = (-1)^{f(x)} |x\rangle \tag{3.13}$$

where

$$U_f : \begin{cases} |w\rangle \to -|w\rangle \\ |x\rangle \to |x\rangle \qquad \forall x \neq w \end{cases} \tag{3.14}$$

Then

$$U_f = 1 - 2|w\rangle\langle w| \tag{3.15}$$

Then

$$U_f = 1 - 2|w\rangle\langle w| \tag{3.16}$$

and

$$U_{f_0} : \begin{cases} |0\rangle \to |0\rangle^{\otimes n} \\ |x\rangle \to -|x\rangle \quad \forall x \neq 00...000 \end{cases} \tag{3.17}$$

Then

$$U_{f_0} = 2|0\rangle\langle 0|^{\otimes n} - I \tag{3.18}$$

The Grover iteration

$$G = (2|\psi\rangle\langle\psi| - I)O \tag{3.19}$$

We have next circuit,

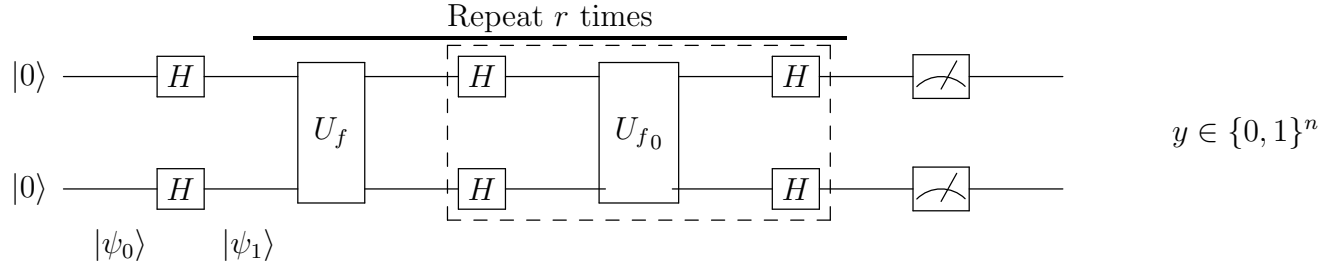


Figure 3.4: Quantum circuit for the Grover algorithm.

Claim: $y = w$

Proof: We define a superposition state

$$|\psi_1\rangle = H^{\otimes n} |0\rangle^{\otimes n} = 2^{-\frac{n}{2}} \sum_{x \in \{0,1\}}^{2^n - 1} |x\rangle \tag{3.20}$$

and (note the dashed line in the circuit)

$$\begin{aligned} V = H^{\otimes n} U_{f_0} H^{\otimes n} &= H^{\otimes n}\left(2\left|0\right\rangle\left\langle 0\right|^{\otimes n} - I\right) H^{\otimes n} \\ &= 2H^{\otimes n}\left|0\right\rangle\left\langle 0\right|^{\otimes n} H^{\otimes n} - H^{\otimes n} I H^{\otimes n} \\ &= 2\left|\psi_1\right\rangle\left\langle\psi_1\right| - I \end{aligned} \tag{3.21}$$

where we used eq. (3.18) and eq. (3.20).

Process: This algorithm carries out the operation $(VU_f)^r$ on the state $\left|\psi_1\right\rangle$.

We use next geometric visualization,
FIG- ¿ Go to notes
Let the circuit (fig. 3.4) us describe,

1. Look at the geometric visualization

2. See the protocol

   - We define

$$\left|\psi\right\rangle = \sqrt{\frac{2^n - 1}{2^n}}\left|w^\dagger\right\rangle + \sqrt{\frac{1}{2^n}}\left|w^\dagger\right\rangle \tag{3.22}$$

   where first terms are not solutions and second are the summation of all solutions. If we define $\sin\theta/2 = 2^{-n/2}$

   - Prepare $\left|\psi\right\rangle$
   - Apply $U_f = I - 2\left|w\right\rangle\left\langle w\right| \rightarrow$ reflection to $\left|w^\dagger\right\rangle$.
   - Apply $V = 2\left|\psi\right\rangle\left\langle\psi\right| - I \rightarrow$ reflection to $\left|\psi\right\rangle$. VU_f is a rotation by an angle θ.
   - After r calls to the oracle , we obtain,

$$p(w) \geq 1 - \sin^2(\theta/2) = 1 - 2^{-n} = \cos^2(\theta/2) \tag{3.23}$$

   -

In this part we can obtain, we see after r applications of the algorithm the state is rotated by $r\theta$,

$$\begin{aligned} r\theta + \frac{\theta}{2} &= \frac{\pi}{2} \\ r &= \frac{\pi}{2\theta} - \frac{\theta}{2\theta} \\ r &= \frac{\pi}{4\arcsin(2^{-n/2})} - \frac{1}{2} \end{aligned}$$

If $n \rightarrow \infty$ then $r = \mathcal{O}(\sqrt{n})$

3.6 Shor's algorithm

Integer factorization problem

Input: An integer N.

Problem: Output positive integers $p_1, p_2, \ldots, p_l, r_1, r_2, \ldots, r_l$ where the p_i are distinct primes and $N = p_1^{r_1} p_2^{r_2} \ldots p_l^{r_l}$ [1, pag. 132].

In 1994 when P. Shor, working for Bell Labs, proposed an algorithm for factoring large numbers on a quantum computer. This algorithm was important for the computer scientists. And it was interesting for the quantum cryptography, since on this area we want efficient methods to factoring large numbers.
While it has not been proven that factoring large numbers can not be archived on a classical computer in polynomial time, as of 2015 the fastest algorithm publicly available for factoring large number runs in

$$\mathcal{O}(e^{\frac{64}{9} n^{1/3} (\log n)^{2/3}},$$

operations where n is the number of bits used to represent the number: this runtime exceeds polynomial time. In contrast Shor's algorithm runs in

$$\mathcal{O}((\log n)^2 \log\log n)$$

on a quantum computer, and then must perform $\mathcal{O}(\log n)$ steps of post processing on a classical computer. Overall then this time is polynomial. This discovery propelled the study of quantum computing forward, as such an algorithm is much sought after.

- This algorithm uses parallelism [3, pag. 163] to propduce a superposition of all values of this function in one step.
- After QFT to create a state in which most of the amplitde is in states close to multiples of reciprocal of the period.
- With high probability, measuring the state yields information from which, by classical means, the period can be extracted. The period is then used to factor M

The function

$$f(r) = x^r \mod n \tag{3.24}$$

is a periodic function, where x is an integer coprime[2] to n. In the context of Shor's algorithm n will be the number we wish to factor. Calculating this function for an exponential number of a's would take exponential time on a classical computer. Shor's algorithm utilizes quantum parallelism to perform the exponential number of operations in one step.

[2]Relatively prime or mutually prime if the only positive integer that is a divisor of both of them is 1. In other words, two numbers are coprime it means that their greatest common divisor is 1. Or Two integers are relatively prime if they share no prime factors

- We use this function to factorize large numbers because
- since $f(a)$ is periodic, (r is the period)

Therefore

$$\begin{aligned} x^r &= 1 \mod n \\ x^{(r/2)2} = x^r &= 1 \mod n \\ x^{(r/2)2} - 1 &= 0 \mod n \\ (x^{r/2} - 1)(x^{r/2} + 1) &= 0 \mod n \end{aligned} \tag{3.25}$$

Eq. (3.25) is true if only if

$$x^r = 1 \mod n \tag{3.26}$$

and r is even. Then we obtain eq. (3.24).
Now the task is factoring n

- randomly choose an integer x and determine the period r of eq. (3.24).

Shor's algorithm

Shor's quantum algorithm attacks the problem of efficiently finding the period of a function. In other words, we use order finding to find the factors of some odd integer N.

Shor's algorithm was fundamental in demonstrating the power and importance of quantum computation. This is an algorithm that can be used to factor prime numbers - meaning that it can be used to break encryption codes if a practical quantum computer is ever built. Needless to say, this algorithm got the attention of a lot of people [2, pag. 216].
The first thing we need to know in order to do Shor's algorithm is order finding . Let x and N be positive integers with no common factors such that $x < N$. The order of x is the smallest positive integer r such that

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

(9.50)

Example 3 *what* mod N *means? First of all,* x *and* N *can't have any common factors because their greatest common divisor is 1.*
Suppose that we let $x = 5$ *and* $N = 44$*. To find* $x^r = a \bmod N$*, we compute* x^r *and subtract* N *until we get the last integer greater than* 0*. The first two cases are less than* $N = 44$*, so we don't do anything:* $5^1 = 5$ *and* $5^2 = 25$*. Now since* $5^3 = 125$*, we note that* $(44)(2) = 88$ *and* $125 - 88 = 37$*. Hence* $5^3 = 37(\mathrm{mod}44)$*. Next* $5^4 = 625$*. We have* $(14)(44) = 616$*, and so* $5^4 = 9(\mathrm{mod}44)$*. Finally* $5^5 = 3125$*. It turns out that* $71 \times 44 = 3124$*, which is 1 less than* $5^5 = 3125$*. This is where we stop. Hence*

$$5^5 = 1(\mathrm{mod}44)$$

The order of 5 is 5 in this case. As you can see, plugging away like this, finding the powers $x^r = 1(\mathrm{mod}N)$ *can be very time-consuming. With large numbers it will swamp the best computers available, the time required is exponential in* $\log N$*. This problem can be solved far more efficiently by using a quantum algorithm based on phase estimation.*

Objective is to factorize the prime numbers.

1. Randomly choose an integer x such that $0 < x < N$. Use the Euclidean algorithm to determine whether x and N are relatively prime. If not, we have found a factor of M. Otherwise, apply the rest of the algorithm.

2. Use quantum parallelism to compute $f(x) = x^r \mod N$ on the superposition of inputs, and apply a quantum Fourier transform to the result.

3. Measure. With high probability, a value r close to a multiple of $\frac{2^n}{r}$ will be obtained.

4. Use classical methods to obtain a conjectured period r from the value v.

5. When r is even, use the Euclidean algorithm to check efficiently whether $a^{r/2} + 1$ (or $a^{q/2} - 1$) has a nontrivial common factor with N.

6. Repeat all steps if necessary[3, pag. 164].

3.7 Letter Grade Distribution

Final letter grades will be assigned at the discretion of the instructor, but here is a minimum guideline for letter grades:

≥ 90.00	A	70.00 - 72.99	C
85.00 - 89.99	A-	67.00 - 69.99	C-
82.00 - 84.99	B+	64.00 - 66.99	D+
79.00 - 81.99	B	60.00 - 63.99	D
76.00 - 78.99	B-	≤ 59.99	F
73.00 - 75.99	C+		

Bibliography

[1] Phillip Kaye, Raymond Laflamme, and Michelle Mosca. *An introduction to quantum computing*. Oxford Univ. Press, 2007.

[2] David McMahon. *Quantum computing explained*. John Wiley & Sons, 2007.

[3] Eleanor Rieffel and Wolfgang Polak. An introduction to quantum computing for non-physicists. *ACM Comput. Surv.*, 32(3):300–335, September 2000.