Crack the Password using the Grover’s Algorithm

EE 4575 Electronics for Quantum Computing Project

for any question, contact: n.khammassi@tudelft.nl

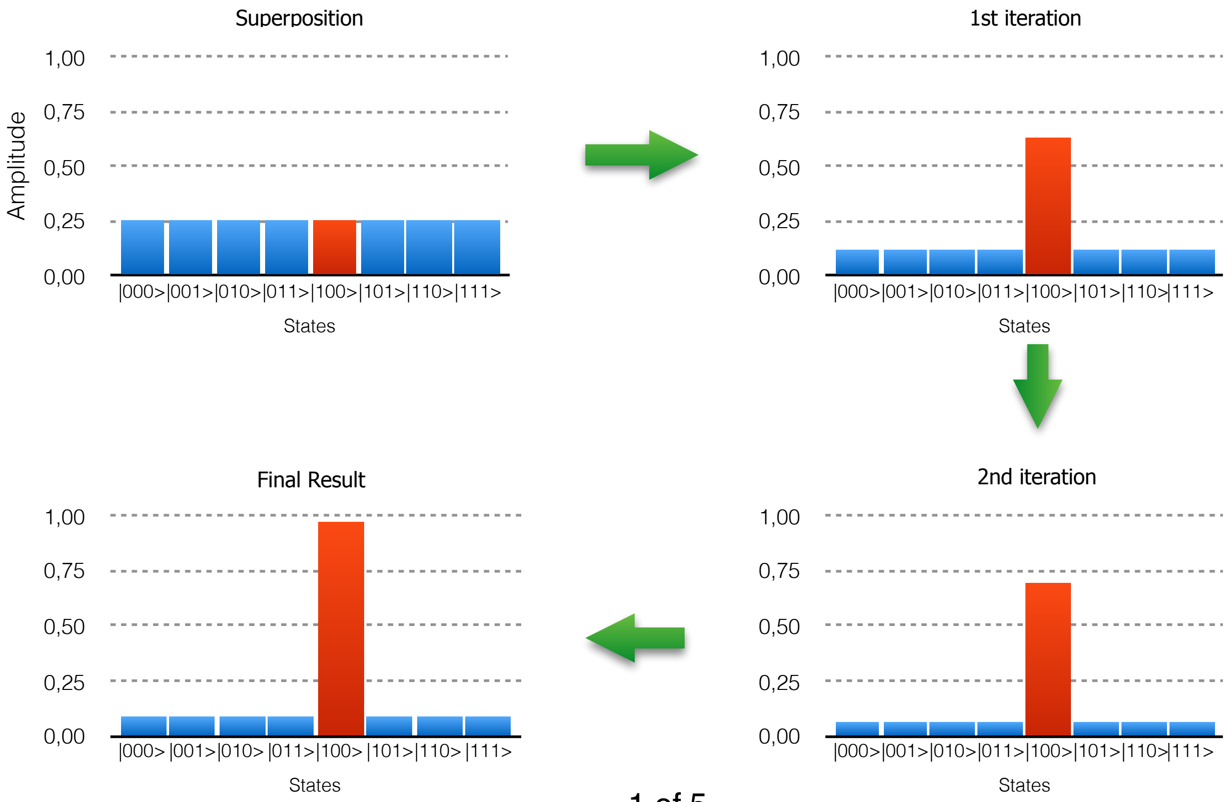
INTRODUCTION

Fast search algorithm is critical to solve many problems which can be formulated as a database searching process. In classical computation, unsorted database search cannot be solved in fewer than O(N) evaluations since the searched element can be the last member of an unsorted search space (worst case).

The Grover’s algorithm is a quantum algorithm which can find with high probability an element of an unordered data set containing N elements using only $O(N^{1/2})$ evaluations. Thus, it provides quadric speedup over its classical counterparts. For instance Grover’s algorithm can brute force a 128-bits symmetric cryptographic key within 2^{64} iterations with a high probability of success while a classical brut force algorithm may requires up to 2^{128} evaluations.

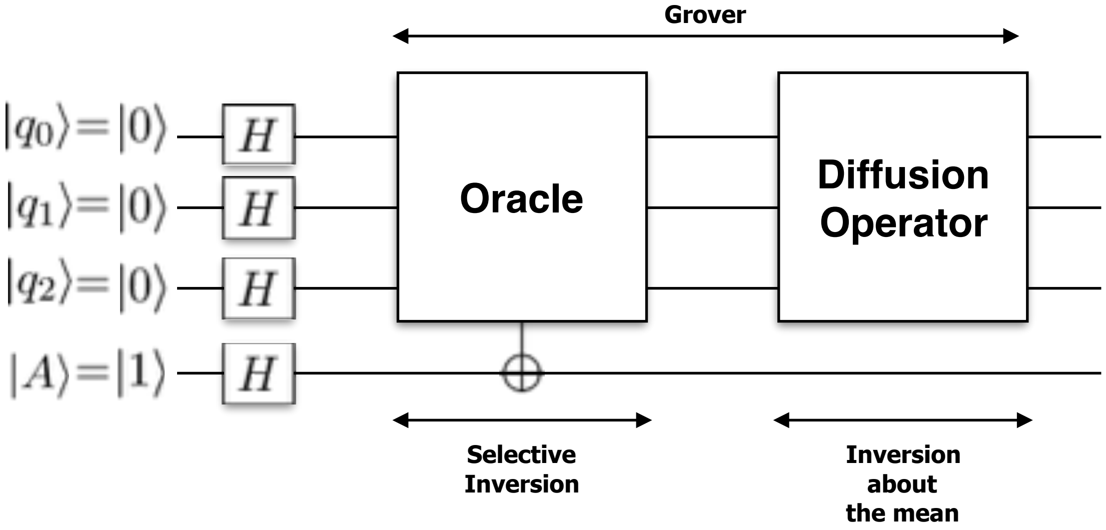
The idea of the Grover’s algorithm is to put a quantum register in an equal superposition state, use an Oracle to *selectively invert the phase* of the searched element then apply an *inversion-about-mean* to amplify the amplitude of the target element. The process is repeated $N^{1/2}$ times for an N elements search space. At the end of the process, we obtain a high probability to measure the target value. It has been proven that no quantum algorithm can solve this search problem in fewer steps than $N^{1/2}$, thus Grover’s algorithm is asymptotically optimal.

Example: Let’s consider a search space of size N=8 containing the elements {0,1,2,…,7} and let (**x***=4) be the element we are looking for. The following figures trace the evolution of the quantum state amplitudes before, during and after applying two ($N^{1/2}$) iterations of the Grover’s algorithm. Each iteration includes applying an Oracle which marks **x*** and the *Diffusion Operator* which invert-about-mean the quantum state amplitudes. We can observe through the result that the target state (**x***=4) is measured with a probability as high as 97%.



THE GROVER'S QUANTUM CIRCUIT

Let's consider the previous example of 3-bits search space. The following figure depict the quantum circuit implementing the Grover's algorithm. The used qubits includes 3 qubits **q_0**, **q_1** and **q_2** used to encode all the possible values of our search space and an auxiliary qubit **A** used to mark the solution. We note that the oracle might use additional auxiliary or scratch qubits to perform its computation.



The Quantum Oracle

Theoretically, the Oracle is only a function which determine whether a value **x** is the target value **x*** we are looking for. Due to linearity of quantum mechanics, when the Oracle circuit is applied to the superposition state, all the possible values are checked simultaneously.

The oracle uses a function f(x) to check if **x** is the target **x*** or not. The function f(x) can be defined as follow:

$$f(x) = 1 \text{ if } x = x^*$$
$$f(x) = 0 \text{ if } x \neq x^*$$

The Oracle flip the auxiliary qubit A when **x** = **x*** using the following operation:

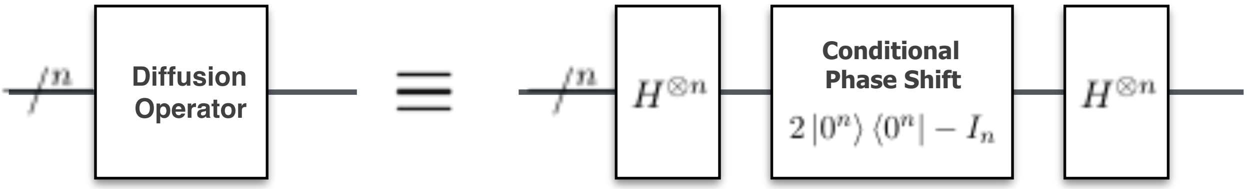
$$|x\rangle\,|A\rangle \rightarrow |x\rangle\,|A \oplus f(x)\rangle$$

The Oracle is a black-box which encode your evaluation function f(x) or simply recognise the solution to the search problem. Thus is depends on what you are looking for !

The Diffusion Operator

The *Diffusion Operator* performs the inversion-about-mean operation which amplifies the amplitude of the target value. At the opposite of the Oracle, this part of the circuit does not depend on the search criteria.

The diffusion operator is mainly composed of the steps depicted on the following block diagram:



The *Conditional Phase Phase Shift* performs mainly the following operation:

$$|x\rangle \rightarrow |0\rangle \quad \text{if } |x\rangle = |0\rangle$$
$$|x\rangle \rightarrow -|x\rangle \text{ if } |x\rangle \neq |0\rangle$$

More details about this block can be found in the literature.

Problem 1 : Search your Number

We want to implement Grover's algorithm to find a number "**x**" within a **6-bits** search space (64 elements).

Let "**s**" be your student number. The number "**x**" we want to find is :

***x =** s **mod** 64*

Write the quantum circuit of the Grover's algorithm to find "**x**" within a 6-bits search space.

The following steps should be followed:

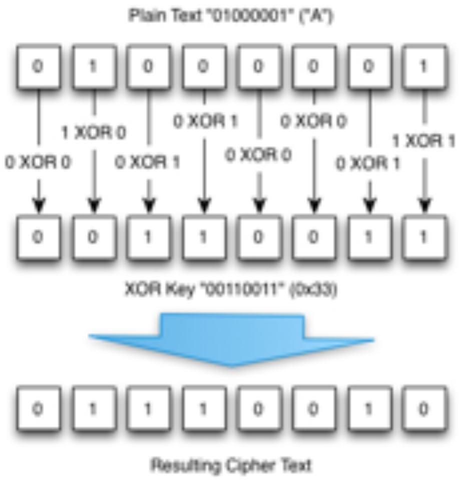
1. Write the quantum Oracle circuit in a separate quantum code file named "***oracle_x.qc***". Test your oracle and check its result before moving to the next step.

2. Write the complete circuit including in another file named "***grover_x.qc***": start by writing a single iteration of the circuit and test it. Then determine the number of required Grover iterations and implement it on your circuit (you can use the loop construct of the Quantum Code, for more information refer to the Annex of this document). Only the final circuit should be delivered.

3. In your report, describe the different Grover's algorithm blocks. In particular, describe how you designed the *Oracle* and the *Diffusion Operator* . Include the graphs of the different quantum state amplitudes you obtained across the different iterations of the circuit.

Problem 2 : Crack the Password !

We have a PDF file which has been encrypted using a simple symmetric encryption algorithm which consists into applying a bitwise XOR operation between each byte of the file and an 8 bits (1 byte) key.



XOR Encryption

To decrypt the file we need to apply the same XOR operations to the encyted file. However, we don't know the secret 8-bits password and we want it to find it !
To find the key we can use brut force and test all the possible 8-bits keys, but how could we know if the key is correct or if we succeeded to decrypt the file if we don't know its content ? We need at least to know one byte of the expected decrypted file.

Our starting point is an important information : the original file is a PDF file. PDF files have a specific structure and have a fixed signature in their header (the first bytes of the file). PDF files starts with the following byte sequence: "%PDF". So we can use the first byte '%' as our reference to test if the password we are testing successfully decrypted the file. Once the password found, the complete file can be decrypted using that same key.

↓

%	P	D	F	…	…	…
…	…	…	…	…	…	…
…	…	…	…	…	…	…
…	…	…	…	…	…	…
…	…	…	…	…	…	…

The ASCII character '**%**' corresponds to the decimal value $(\mathbf{37})_{10}$ and the binary value $(\mathbf{00100101})_{2}$.

The first byte of our **encrypted file** is $(\mathbf{1101110})_{2}$ which correspond to the ASCII character '**n**'.

Instead of doing a naive classical brut force, we want to use the Grover's quantum algorithm to find the 8-bits password.

1. Write the Grover's quantum algorithm which can find the secret key ! Save the circuit in a file named "***grover_crack.qc***".

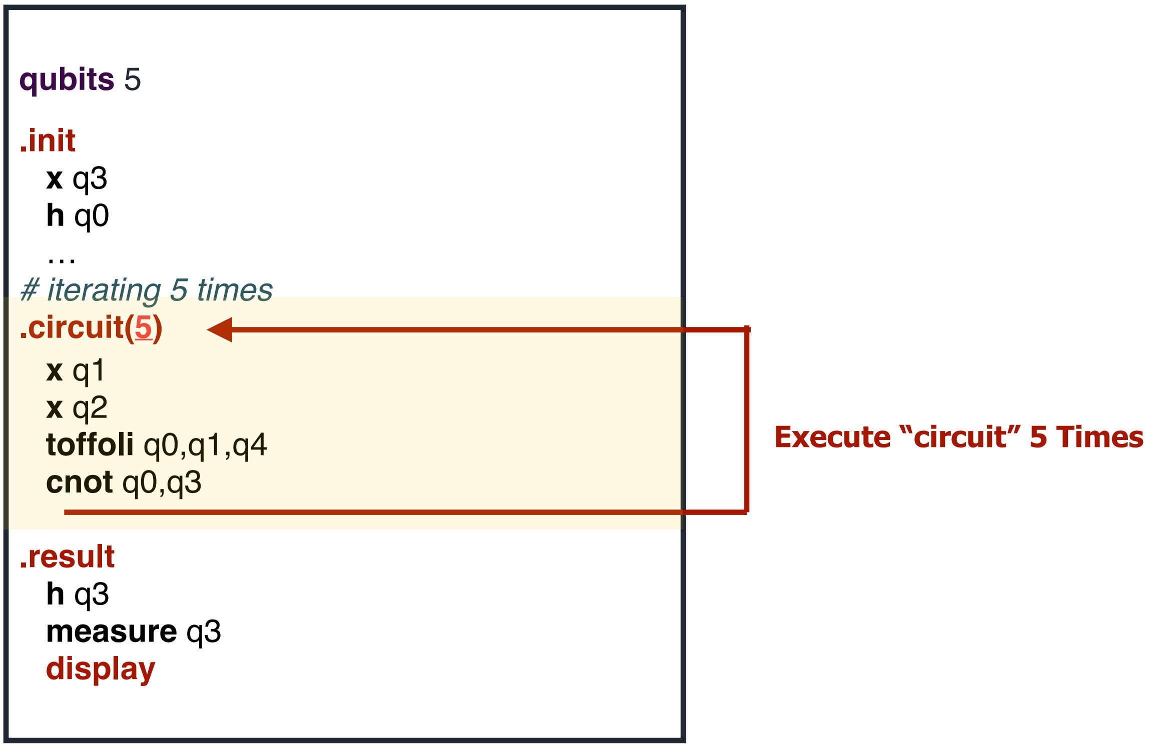
2. We saw during the different lectures that a "**qubit**" is an expensive resource both on the physical platform (limited number of qubits) and the universal simulation platform (memory requirement) ! We want to use the smallest possible number of qubits. Optimise your original circuit to use lesser qubits and save the optimised version in a file named "**grover_crack_optimized.qc**"

3. In your report, describe how you designed the different blocks of your circuits and how you optimised the initial version to reduce the number of qubits. Don't forget to precise what is the secret key !

4. During your work on this project, you might need to find out more information or details about Grover's algorithm or the different blocks composing the circuit. You can give a list of references at the end of your report.

ANNEX - LOOPS IN THE QUANTUM CODE

If we want to execute a sub-circuit named "circuit" 5 times, we can simply add the number of iterations between parenthesis to the name of the sub-circuit. As a result, the QX Simulator will execute that sub-circuit 5 times as shown in the following figure.



```
qubits 5

.init
   x q3
   h q0
   …
# iterating 5 times
.circuit(5)
   x q1
   x q2
   toffoli q0,q1,q4
   cnot q0,q3

.result
   h q3
   measure q3
   display
```

Execute "circuit" 5 Times