Quantum Search

Miguel Sozinho Ramalho

November, 2018

Table of contents

Introduction

This week we will be studying a famous and pragmatic quantum algorithm to perform a search, the **Grover's algorithm**. We will look through its purpose, how it compares with classical search strategies that solve the same problems. The algorithm will be studied from a more abstract point of view, namely in understanding **phase inversion** and **inversion about the mean**. This week's exercises will focus more on the implementation details.

The Search Problem

Let us assume we have a function f:

$$f : \{0, 1, ..., N-1\} \rightarrow \{0, 1\}$$

And we want to find those values of x for which $f(x) = 1$. To simplify, and also to make the problem harder, let us assume there is only one such x:

x	f(x)
0	0
1	0
...	0
i	1
...	0
N	0

The Search Problem

How would a classical programmer tackle this problem?

- Sequential search through $x \in [0, N[$ until $f(x) = 1$
- Random search with $x \in [0, N[$ until $f(x) = 1$
- ...

As you can see, this is quite straightforward, and yields a worst-case scenario of N searches and average $\frac{N}{2}$ (assuming no bias in f).

SAT Problem

Besides the most obvious problems, we should also know about the **SAT problem** (sometimes Boolean satisfiability problem). This problem is about determining values for Boolean variables so that the **Boolean expression** associated **evaluates to true**. This is a computationally expensive problem and falls into the **NP** (actually NP-complete) category. Many problems, like scheduling, can be converted into a SAT formulation, making use of existing SAT solvers to easily find solutions for novel problems.

As such, SAT can be seen as a **search problem**, where we need to find the precise combination of Boolean values that **yields true**.

Grover's algorithm

Luckily for the world, Dr. Lov Grover devised an algorithm that, instead of N steps to find a solution, requires $\sqrt{N}$. This is a it is optimal (no better speed up is possible for this problem).

The algorithm makes use of two very important phenomenon that can be implemented in a quantum computer:

- **Phase inversion**
- **Inversion about the mean** (average)
- An oracle for our f function

Phase Inversion

The first phenomena is that of phase inversion and will help in finding x' so that $f(x') = 1$. Let us imagine a quantum superposition over all possible x values:

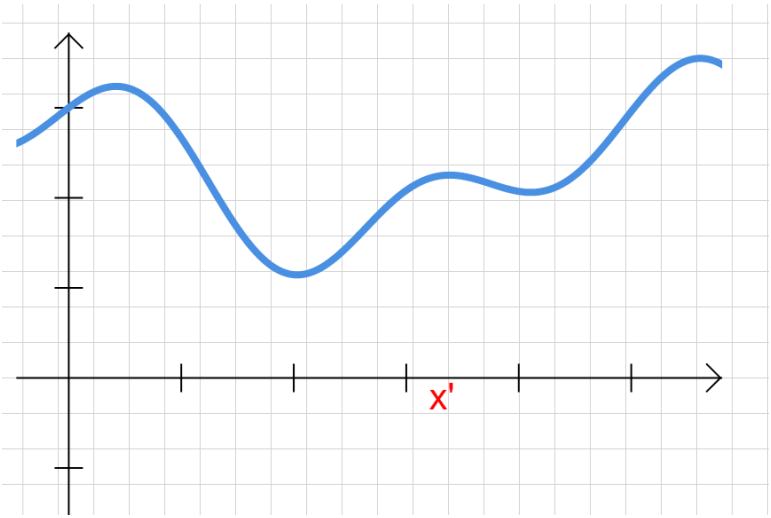
$$\sum_x \alpha_x |x\rangle$$

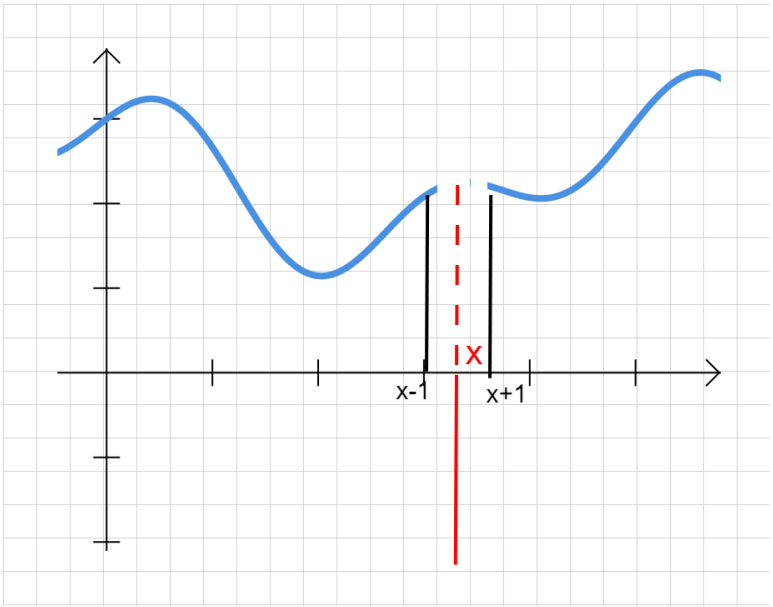
To which we can, through the use of our oracle, obtain:

$$\sum_{x \neq x'} \alpha_x |x\rangle - \alpha_{x'} |x'\rangle$$

Phase Inversion

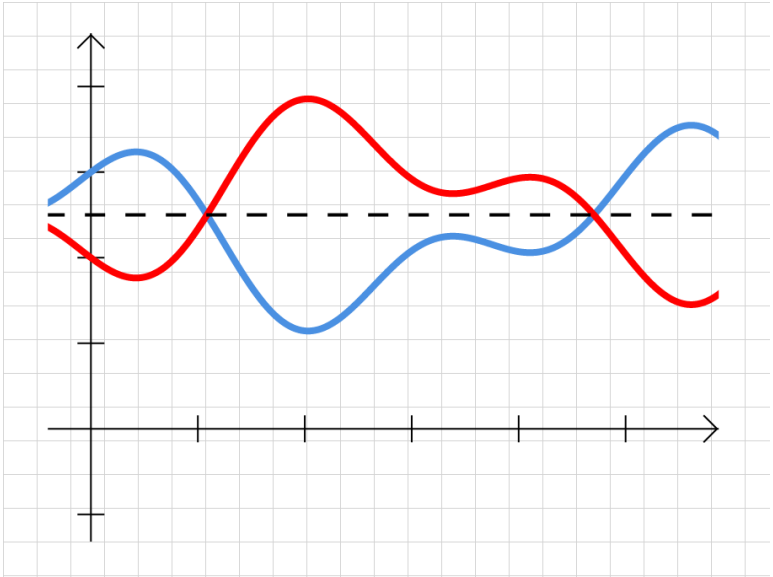
This will mean that, if we plot α_x for each x value, we will get something like the following abstract diagram - in which the real value has been inverted (even though we do not know its value)





Inversion about the Mean

The next phase of the algorithm is to take that very distribution of weight and mirror it to the other side of the mean. This can be done in a quantum circuit (more on the exercises). The following is a representation of this transformation (either function is the inversion about the mean of the other):



Inversion about the Mean

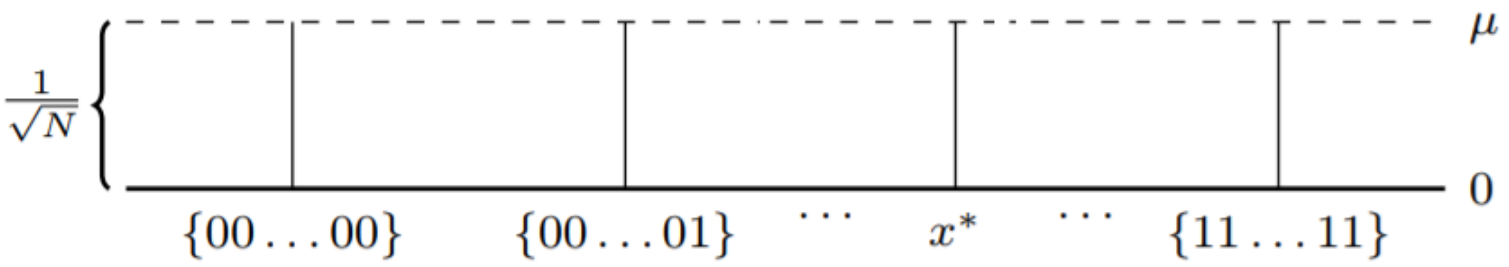
Why do we do this, you may ask. To answer this, consider the two steps taken so far: phase inversion and then . The first one, sets our solution to be a bit more recognizable and the second operation further segregates our result from the rest of the possible states.

If we keep on doing this two steps, we will segregate the correct answer from the others further and further. The number of times we need to do it to have a guaranteed high probability of having the right answer is $\sqrt{N}!$

Grover's algorithm (why $\sqrt{N}$?)

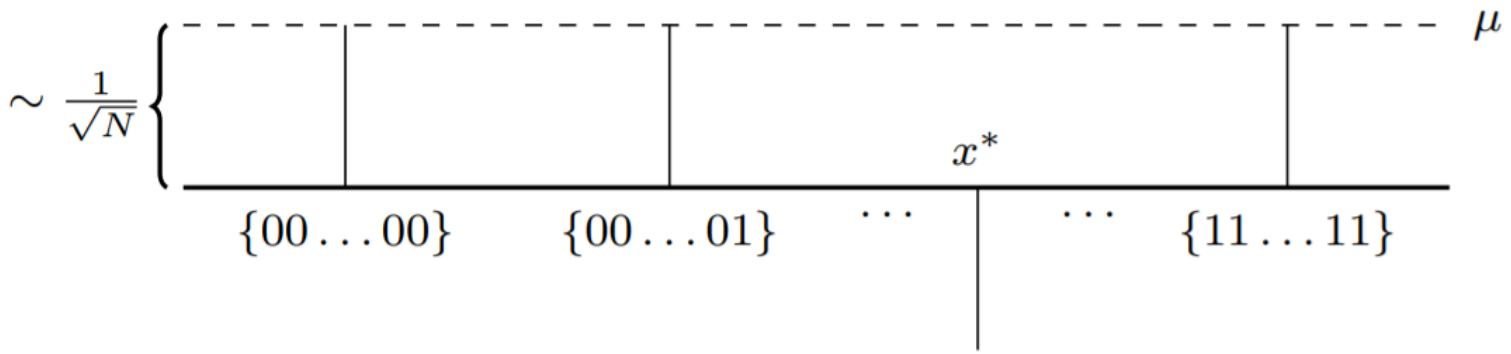
Let us assume, we initialize our n qubits to $|0\rangle$, each qubit can be compared to a Boolean value on the n-bit solution we are looking for. We then create a superposition that sets them in a **uniform superposition** (each state has equal probability):

$$\sum_{x \in 0,1^n} \frac{1}{\sqrt{N}} |x\rangle$$



Grover's algorithm (why $\sqrt{N}$?)

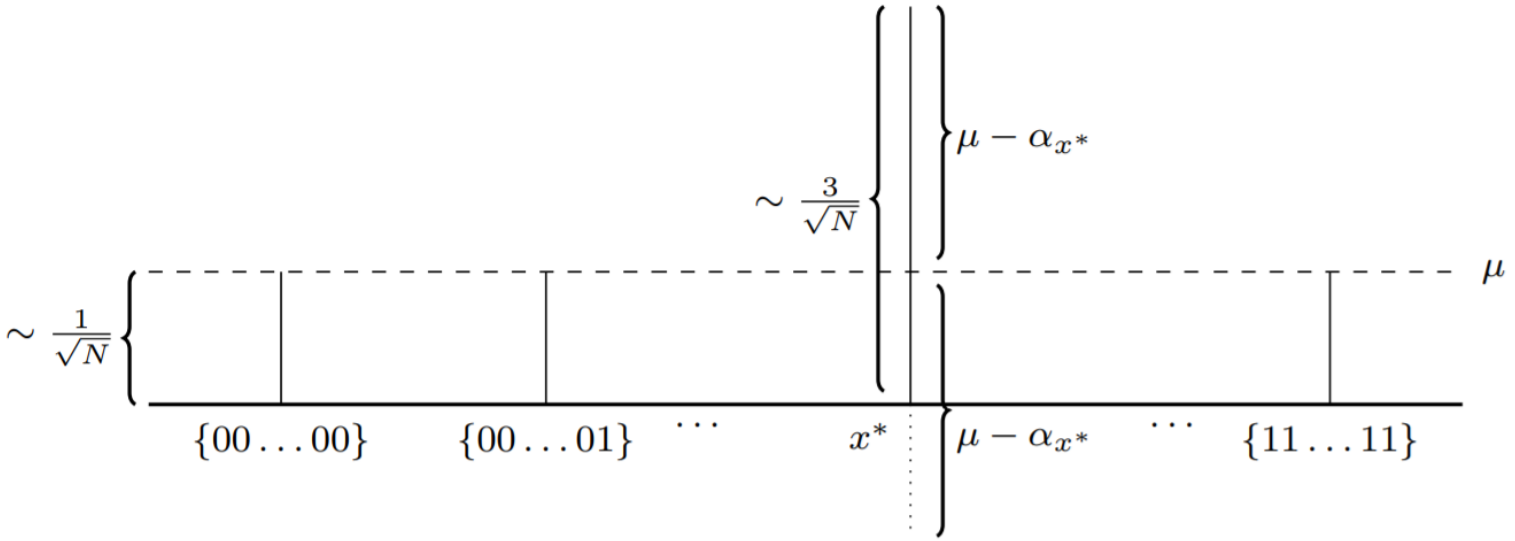
We then create a quantum oracle O_f that will flip the qubits if the result is 1, and apply this gate, this effectively applies a phase inversion:



Grover's algorithm (why $\sqrt{N}$?)

Finally, we will apply the ($\mu = \frac{1}{N}\sum_x \alpha_x |x\rangle$):

$$\sum_x \alpha_x |x\rangle \to \sum_x (2\mu - \alpha_x)|x\rangle$$



Grover's algorithm ($\sqrt{N}$ because...)

If you repeat this mechanism, you will notice the amplitude improves (per step) $\frac{\sqrt{2}}{\sqrt{N}}$ every iteration:

$$\frac{1}{\sqrt{N}}, \frac{3}{\sqrt{N}}, \frac{5}{\sqrt{N}}, \cdots \frac{1}{\sqrt{2}}$$

At the point of $\frac{1}{\sqrt{2}}$ we have a guaranteed high probability of the measurement resulting in the solution! How long until we get there?

$$\frac{\frac{1}{\sqrt{2}}}{\frac{\sqrt{2}}{\sqrt{N}}} = \frac{\sqrt{N}}{2} = O(\sqrt{N})$$

Grover's algorithm

It should be noted that the amplitude **cannot increase forever** and will eventually be so large that the mean becomes negative (after phase inversion) and the step is actually **harming our result**. That is why one should know hen to stop (how many iterations guarantee the minimum desired amplitude).

There you have it, by executing this two-step iteration we can solve a classical problem that had a complexity of $O(N)$ in $O(\sqrt{N})$, it is not an exponential growth (as some quantum algorithms are able to do), but in this case that would have some very disruptive consequences.

Hands-on

This week's exercises will focus more on understanding the logic at the quantum circuit level, and are based on a Jupyter notebook available at the official QISKit tutorial repository. You will get to solve a real SAT problem, of dimension 3, in a real quantum device!

Where to learn more?

- IBM Q Experience Documentation on Grover includes circuitry and interesting diagrams.
- An Introduction to Quantum Computing, Without the Physics good paper for understanding more on Grover's algorithm