Grover search algorithm

Quantum Computing Minicourse ICTP-SAIFR

Stefano Carrazza[‡] and Matteo Robbiati[†]

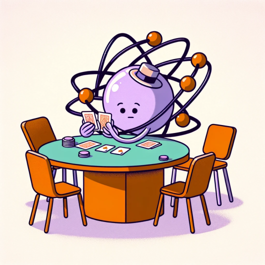
8 April 2024

[‡] Associate Professor & Researcher, University of Milan and INFN Milan, Italy.
[†] PhD candidate, University of Milan, Italy and CERN, Switzerland.

Motivation

The Grover algorithm is powerful when searching an item among an unordered set of candidates.

Extract the jack of clubs from a Poker deck



Find a passcode composed of 10 numbers

Find an antidote to the Cobra poison, exploring 10^{20} molecules

? How many attempts could you need, in the worst scenario, to explore all the possibilities?

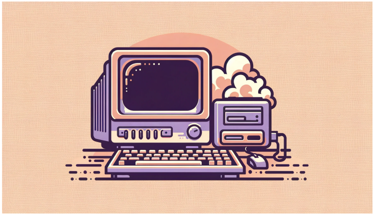
⚠ In the worst scenario, you will need to check 52 **cards**, 10^{10} **passcodes** and 10^{20} **molecules**.

Quadratic speedup

If we consider a time cost of $\delta = 10^{-8}$ seconds for any algorithmic call (quantum or classical) we would wait:

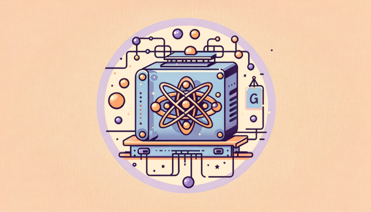
On a classical computer

- 0.52 μs to find the jack of clubs;
- 100 seconds to find the passcode;
- $\sim$ 31688 years to find the Cobra antidote.



On a quantum computer

- 0.0721 μs to find the jack of clubs;
- 0.001 seconds to find the passcode;
- 100 seconds to find the Cobra antidote.



The Grover algorithm solves this kind of search with a number of algorithmic calls proportional to $\sqrt{N}$, where N is the dimension of the search space.

The Grover algorithm

The key steps of the Grover algorithm:

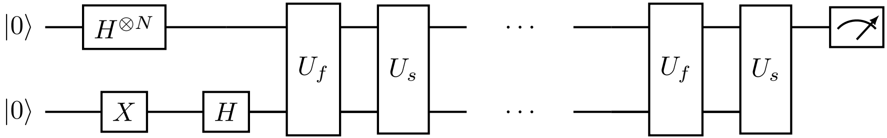
1. prepare a system of N qubits into a maximally superposed state;
2. prepare an ancilla qubit into the $|-\rangle$ state;
3. apply an oracle operator U_f which can mark the correct solution;
4. apply a diffusion operator U_s which amplifies the correct solution;
5. repeat 3. and 4. for the optimal number of times.

The Grover algorithm

The key steps of the Grover algorithm:

1. prepare a system of N qubits into a maximally superposed state;
2. prepare an ancilla qubit into the $|-\rangle$ state;
3. apply an oracle operator U_f which can mark the correct solution;
4. apply a diffusion operator U_s which amplifies the correct solution;
5. repeat 3. and 4. for the optimal number of times.

In terms of quantum circuit:



step 1 and 2: the state preparation

We consider a set of 2^N unordered items and we encode them into the state of an N qubits system:

$$\begin{bmatrix} \text{item}_1 \\ \text{item}_2 \\ \dots \\ \text{item}_{2^N} \end{bmatrix} \qquad \rightarrow \qquad |\psi\rangle = \begin{bmatrix} \psi_{00\dots0} \\ \psi_{00\dots1} \\ \dots \\ \psi_{11\dots1} \end{bmatrix} \equiv \begin{bmatrix} x_0 \\ x_1 \\ \dots \\ x_{2^N-1} \end{bmatrix}.$$

step 1 and 2: the state preparation

We consider a set of 2^N unordered items and we encode them into the state of an N qubits system:

$$\begin{bmatrix} \text{item}_1 \\ \text{item}_2 \\ \dots \\ \text{item}_{2^N} \end{bmatrix} \qquad \rightarrow \qquad |\psi\rangle = \begin{bmatrix} \psi_{00\dots0} \\ \psi_{00\dots1} \\ \dots \\ \psi_{11\dots1} \end{bmatrix} \equiv \begin{bmatrix} x_0 \\ x_1 \\ \dots \\ x_{2^N-1} \end{bmatrix}.$$

We make use of an ancilla qubit too. The state of the system is then: $|x\rangle_N |y\rangle$.

step 1 and 2: the state preparation

We consider a set of 2^N unordered items and we encode them into the state of an N qubits system:

$$\begin{bmatrix} \text{item}_1 \\ \text{item}_2 \\ \dots \\ \text{item}_{2^N} \end{bmatrix} \qquad \rightarrow \qquad |\psi\rangle = \begin{bmatrix} \psi_{00\dots0} \\ \psi_{00\dots1} \\ \dots \\ \psi_{11\dots1} \end{bmatrix} \equiv \begin{bmatrix} x_0 \\ x_1 \\ \dots \\ x_{2^N-1} \end{bmatrix}.$$

We make use of an ancilla qubit too. The state of the system is then: $|x\rangle_N |y\rangle$.

In this framework, the target solution ω is represented by one of the components of $|x\rangle_N$.

step 1 and 2: the state preparation

We consider a set of 2^N unordered items and we encode them into the state of an N qubits system:

$$\begin{bmatrix} \text{item}_1 \\ \text{item}_2 \\ \dots \\ \text{item}_{2^N} \end{bmatrix} \quad \rightarrow \quad |\psi\rangle = \begin{bmatrix} \psi_{00\dots0} \\ \psi_{00\dots1} \\ \dots \\ \psi_{11\dots1} \end{bmatrix} \equiv \begin{bmatrix} x_0 \\ x_1 \\ \dots \\ x_{2^N-1} \end{bmatrix}.$$

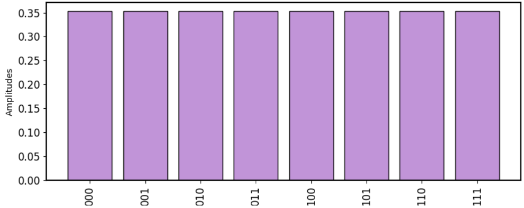
We make use of an ancilla qubit too. The state of the system is then: $|x\rangle_N |y\rangle$.

In this framework, the target solution ω is represented by one of the components of $|x\rangle_N$.

The first step of the algorithm is the state preparation into the following superposed state:

$$H^{\otimes N+1} |0\rangle_N |1\rangle = \left[\frac{1}{2^{N/2}} \sum_{i=0}^{2^N-1} |x_i\rangle \right] \otimes |-\rangle .$$

We move the system state from the computational zero to the maximally superposed state $|s\rangle$.



Amplitudes of $|x\rangle_N$ with $N = 3$ and $|\omega\rangle = |001\rangle$

Step 3: the oracle U_f

We consider now a function $f : \{0,1\}^N \to \{0,1\}$ which can detect the correct solution $|\omega\rangle$:

$$f(x) = \begin{cases} 1 & \text{if } x = \omega, \\ 0 & \text{otherwise.} \end{cases}$$

Step 3: the oracle U_f

We consider now a function $f : \{0,1\}^N \to \{0,1\}$ which can detect the correct solution $|\omega\rangle$:

$$f(x) = \begin{cases} 1 & \text{if } x = \omega, \\ 0 & \text{otherwise.} \end{cases}$$

This function is typically embedded into a quantum oracle U_f which is able to recognize the solution for us. It's important to underline that even if the oracle can detect the solution, may don't know it's exact value.

Step 3: the oracle U_f

We consider now a function $f : \{0,1\}^N \to \{0,1\}$ which can detect the correct solution $|\omega\rangle$:

$$f(x) = \begin{cases} 1 & \text{if } x = \omega, \\ 0 & \text{otherwise.} \end{cases}$$

This function is typically embedded into a quantum oracle U_f which is able to recognize the solution for us. It's important to underline that even if the oracle can detect the solution, may don't know it's exact value.

The oracle marks the solution (one of the amplitudes) by flipping its sign thanks to a phase-kickback procedure.

Step 3: the oracle U_f

We consider now a function $f : \{0,1\}^N \to \{0,1\}$ which can detect the correct solution $|\omega\rangle$:

$$f(x) = \begin{cases} 1 & \text{if } x = \omega, \\ 0 & \text{otherwise.} \end{cases}$$

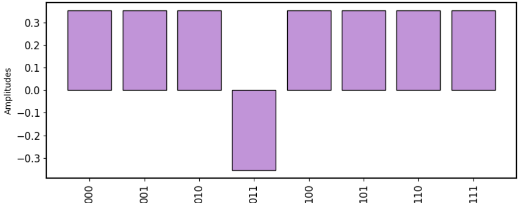
This function is typically embedded into a quantum oracle U_f which is able to recognize the solution for us. It's important to underline that even if the oracle can detect the solution, may don't know it's exact value.

The oracle marks the solution (one of the amplitudes) by flipping its sign thanks to a phase-kickback procedure.

In practice, this can be done by setting up a multi-controlled operation which applies a phase kickback only if the control state is $|\omega\rangle$.

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

where $f(x)$ follows the rule exposed before.



Amplitudes of $|x\rangle_N$ with $N = 3$ and $|\omega\rangle = |001\rangle$

Step 4: the diffusion operator U_s

Now $|\omega\rangle$ is marked. We need to amplify its amplitude so that we can detect it among the 2^N components of $|x\rangle$.

Step 4: the diffusion operator U_s

Now $|\omega\rangle$ is marked. We need to amplify its amplitude so that we can detect it among the 2^N components of $|x\rangle$.

To this purpose, we introduce the diffusion operator

$$U_s = 2\,|s\rangle\,\langle s| - I,$$

whose action is a reflection of the system with respect to the state:

$$|s\rangle = \frac{1}{2^{N/2}} \sum_{i=0}^{2^N-1} |x_i\rangle\,.$$

Step 4: the diffusion operator U_s

Now $|\omega\rangle$ is marked. We need to amplify its amplitude so that we can detect it among the 2^N components of $|x\rangle$.

To this purpose, we introduce the diffusion operator

$$U_s = 2\,|s\rangle\,\langle s| - I,$$

whose action is a reflection of the system with respect to the state:

$$|s\rangle = \frac{1}{2^{N/2}} \sum_{i=0}^{2^N-1} |x_i\rangle\,.$$

Since the amplitude of $|\omega\rangle$ was flipped by U_f, the action of U_s will produce an increasing of $|\omega\rangle$ and, consequently, a decreasing of the other components.

Step 4: the diffusion operator U_s

Now $|\omega\rangle$ is marked. We need to amplify its amplitude so that we can detect it among the 2^N components of $|x\rangle$.

To this purpose, we introduce the diffusion operator

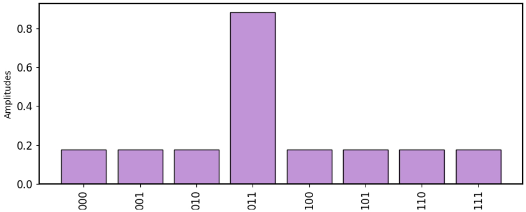
$$U_s = 2\,|s\rangle\,\langle s| - I,$$

whose action is a reflection of the system with respect to the state:

$$|s\rangle = \frac{1}{2^{N/2}} \sum_{i=0}^{2^N-1} |x_i\rangle\,.$$

Since the amplitude of $|\omega\rangle$ was flipped by U_f, the action of U_s will produce an increasing of $|\omega\rangle$ and, consequently, a decreasing of the other components.

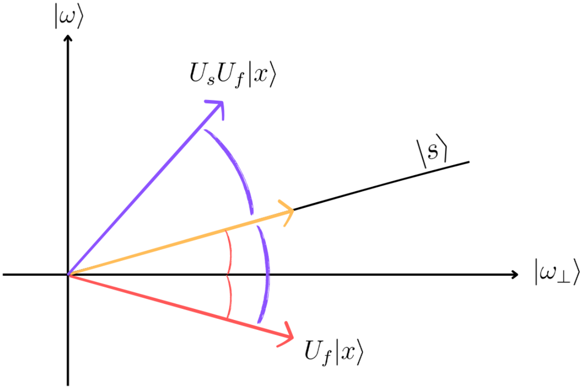
U_s is also known as "inversion by the mean", in fact, it can be shown it implements an inversion w.r.t. the mean value of the amplitudes of $|x\rangle$.



Amplitudes of $|x\rangle_N$ with $N = 3$ and $|\omega\rangle = |001\rangle$

Graphical intuition

We can visualize the Grover's action using vectors.



We are iteratively moving the system state through the target $|\omega\rangle$.

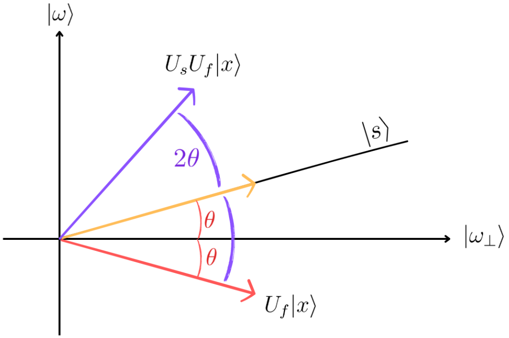
How many times do we need to iterate Grover?

As we can deduce from the previous slide, there exist an **optimal number** of Grover iterations fixed by geometry.

1. we can decompose $|x\rangle$ into the *winning* and the *losing* components $|s\rangle = \sqrt{\frac{1}{N}}\,|\omega\rangle + \sqrt{\frac{N-1}{N}}\,|\omega_\perp\rangle$.
2. The same vector can be defined in terms of the angle in the plane: $|s\rangle = \sin\theta\,|\omega\rangle + cos\theta\,|\omega_\perp\rangle$.
3. from 1. and 2. we can write $\theta = \arcsin\left(1/\sqrt{N}\right)$ and, if N is large, $\theta \approx 1/\sqrt{N}$.
4. the action of $U_s U_f$ on $|x\rangle$ is equal to a rotation of 2θ of the vector.
5. after k iteration of Grover, the angle has become: $\alpha = (2k+1)\,\theta$, and, to maximize $\sin\alpha$:

$$\alpha = \frac{\pi}{2} \quad \rightarrow \quad k = \frac{\pi}{4\theta} - \frac{1}{2} = \frac{\pi}{4}\sqrt{N} - \frac{1}{2}.$$

6. from 5. we need to get an integer, since we are talking about iterations. Commonly $\theta \approx \frac{\pi}{4}\sqrt{N}$.



Let's code!