*Consiglio Nazionale delle Ricerche*
*Istituto di Calcolo e Reti ad Alte Prestazioni*

# Quantum Algorithm for the Boolean Satisfiability Problem

Riccardo Rende[1], Carlo Mastroianni[2], Francesco Plastina [1,3]

**RT-ICAR-CS-19-06** **October 2019**

1) Unical, Dip.di Fisica, Via P. Bucci 30 C, 87036 Rende (CS), email: francesco.plastina@fis.unical.it, rendericcardo@gmail.com
2) ICAR-CNR, Via P. Bucci 8/9 C, Rende (CS), email: carlo.mastroianni@icar.cnr.it
3) INFN – Gruppo Collegato di Cosenza

Consiglio Nazionale delle Ricerche, Istituto di Calcolo e Reti ad Alte Prestazioni (ICAR)
– Sede di Cosenza, Via P. Bucci 8-9C, 87036 Rende, Italy, URL: *www.icar.cnr.it*
– Sezione di Napoli, Via P. Castellino 111, 80131 Napoli, URL: *www.icar.cnr.it*
– Sezione di Palermo, Via Ugo La Malfa, 153, 90146 Palermo, URL: *www.icar.cnr.it*

# 1 Abstract

In questo rapporto tecnico faremo una breve introduzione alla computazione quantistica, per poi presentare **l'algoritmo di Grover** ed una sua applicazione. La computazione quantistica si basa sui **qubits**, sistemi quantistici a due livelli che possono trovarsi in sovrapposizioni di stati. Un generico algoritmo quantistico può essere espresso come l'applicazione di una sequenza di **gates**, ossia operatori unitari che agiscono su un qubit o su una coppia di qubits. Grover presenta una soluzione al problema della **ricerca lineare**: abbiamo un **database non strutturato** di N elementi di cui soltanto uno soddisfa una determinata condizione, è questo che dobbiamo trovare. L'algoritmo di Grover introduce uno speed-up quadratico rispetto all'algoritmo classico, che consiste nell'esaminare singolarmente gli elementi nel database. Ciò deriva dalla possibilità dei qubits di stare in sovrapposizione di stati che porta al parallelismo intrinseco della computazione quantistica. Utilizzeremo infine l'algoritmo di Grover per affrontare il problema SAT, cioè trovare la soluzione di una formula logica booleana espressa in forma normale congiuntiva, cioè come una congiunzione di disgiunzioni.

# Indice

# 2 Introduzione

## 2.1 Qubit

Così come il bit è alla base della computazione classica, il qubit (o quantum bit) è alla base della computazione quantistica. Un bit classico può assumere solo due valori che possiamo indicare con 0 e 1; analogamente, un qubit può trovarsi in due stati di base $|0\rangle$ e $|1\rangle$ che possiamo pensare come le configurazioni che rappresentano i valori classici 0 e 1. La novità è che il qubit, essendo un sistema quantistico, può trovarsi anche in una sovrapposizione di stati e dunque possiamo avere

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \tag{1}$$

Possiamo definire, formalmente, un qubit come un sistema a due livelli, il cui stato vive nello spazio

$$\mathscr{H} \cong \mathbb{C}^2 = \mathrm{Span}\{|0\rangle, |1\rangle\} \tag{2}$$

Dunque, possiamo esprimere il generico stato puro $|\psi\rangle \in \mathscr{H}$ nella forma

$$|\psi\rangle = \cos\frac{\theta}{2}|0\rangle + e^{i\varphi}\sin\frac{\theta}{2}|1\rangle \tag{3}$$

con $\theta \in [0, \pi]$ e $\varphi \in [0, 2\pi]$. Infatti, il primo coefficiente può sempre essere scelto reale e, per avere stati normalizzati, prendiamo un seno ed un coseno.

Poiché lo stato è individuato dai parametri $\theta$ e $\varphi$, possiamo associarlo univocamente ad un punto sulla superficie di una sfera unitaria (detta sfera di Bloch) e cioè ad un versore $\hat{n} = (sin\theta cos\varphi, sin\theta sin\varphi, cos\theta)$.

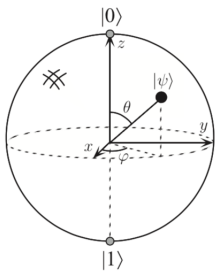



Figura 1: Sfera di Bloch

Fisicamente, un qubit può essere realizzato con qualsiasi sistema quantistico con spazio di Hilbert bidimensionale; come, ad esempio, lo spin di un elettrone o la polarizzazione di un fotone.

Quando per un qubit specifichiamo la base $|0\rangle$ e $|1\rangle$, abbiamo fissato un'osservabile $\hat{O}$ del sistema, i cui autostati sono quelli della base scelta. Anche se in futuro non lo specificheremo più; quando diremo di effettuare una misura sul sistema, intenderemo la misura di tale osservabile. Se lo stato del sistema, prima che si effettui la misura, è $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, in accordo con il postulato della misura, i possibili risultati saranno 0 o 1, rispettivamente con probabilità $|\alpha|^2$ e $|\beta|^2$ con $|\alpha|^2 + |\beta|^2 = 1$ ed il nuovo stato del sistema sarà l'autostato corrispondente al risultato ottenuto.

Supponiamo ora di avere due qubit. Se questi fossero due bit classici, avremmo quattro possibili stati 00, 01, 10, 11. Corrispondentemente, un sistema di due qubit ha quattro componenti di base denotate con $|00\rangle$, $|01\rangle$, $|10\rangle$, $|11\rangle$ e lo stato della coppia può essere una generica combinazione lineare di questi stati di base.

$$|\psi\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle \tag{4}$$

Ora, i possibili risultati di una misura sono 00, 01, 10, 11, rispettivamente con probabilità $|\alpha_{00}|^2$, etc. Lo spazio di Hilbert del sistema totale si ottiene componendo tensorialmente gli spazi di Hilbert dei singoli qubit $\mathscr{H} = \mathscr{H}_1 \otimes \mathscr{H}_2$. La dimensione è, dunque 4 e quella usata per esprimere il generico stato del sistema è solo una delle possibili basi di questo spazio, in particolare è la base fattorizzata.

In generale, per un sistema di $N$ qubits, lo spazio di Hilbert complessivo è $\mathscr{H} = \mathscr{H}_1 \otimes \cdots \otimes \mathscr{H}_N$, con dimensione $2^N$ ed il generico stato, scritto nella base fattorizzata, è esprimibile come

$$|\psi\rangle = \sum_{x=00\ldots0}^{11\ldots1} C_x|x\rangle \tag{5}$$

Il fatto che la dimensione dello spazio di Hilbert scali esponenzialmente con il numero di qubit è uno dei principali punti di forza della computazione quantistica. Questo ci permette di avere stati che sono sovrapposizione di altri $2^N$ stati; il che, come vedremo, è un grande vantaggio. Se volessimo si-

mulare su un computer classico un sistema di $N$ qubit, dovremmo manipolare $2^N$ coefficienti: già per $N = 30$ la simulazione risulterebbe impossibile.

## 2.2 Gate

La computazione classica si basa sull'applicazione di gate logici sui bit; analogamente, la computazione quantistica si basa sull'applicazione di gate quantistici ai qubit, che ne modificano lo stato. Gli operatori $\hat{O}$, definiti sugli stati del sistema, sono la controparte dei gate quantistici nella rappresentazione del sistema nello spazio di Hilbert. Poiché la norma degli stati deve conservarsi, i gate quantistici devono essere delle operazioni unitarie e, dunque, sempre reversibili: è questa una delle principali differenze rispetto alle operazioni classiche.
I gate quantistici possono agire su un singolo qubit, o su molti di essi. Gate ad uno e due qubit, tuttavia, bastano per costruire tutte le possibili operazioni. Di seguito, introdurremo soltanto i più importanti gate che useremo nell'applicare l'algoritmo di Grover. L'azione di un gate quantistico su dei qubit può essere rappresentata come l'applicazione di una matrice sul vettore che ne rappresenta lo stato.

**X (quantum NOT gate):** è un gate che agisce su un solo qubit. Definiamo il gate $X$ specificandone la rappresentazione matriciale

$$X \doteq \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \tag{6}$$

Questo agisce sugli stati di base mandando $|0\rangle$ in $|1\rangle$ e viceversa; per questo motivo, esso viene chiamato NOT quantistico. Essendo lineare, agisce sul generico stato di un qubit come

$$X \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \beta \\ \alpha \end{bmatrix} \tag{7}$$

Notiamo che esso è hermitiano, $X^\dagger = X$, e anche unitario $X^2 = I$.

**Z:** un altro importante gate che agisce su un singolo qubit è

$$Z \doteq \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \tag{8}$$

questo lascia $|0\rangle$ inalterato e cambia il segno ad $|1\rangle$ trasformandolo in $-|1\rangle$.

**H (Hadamard gate):** anch' esso agisce su un singolo qubit ed è definito come

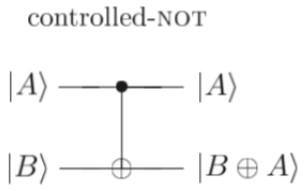
$$H \doteq \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{9}$$

e manda $|0\rangle$ in $(|0\rangle + |1\rangle)/\sqrt{2}$ e $|1\rangle$ in $(|0\rangle - |1\rangle)/\sqrt{2}$. Notiamo che $H^2 = I$.

**CNOT (controlled NOT):** è il più importante gate a due qubit, definito dalla matrice

$$U_{CN} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{10}$$

I due qubit di input sono detti *control* e *target* qubit, rispettivamente. È schematizzato come segue




la linea superiore rappresenta il qubit di controllo, mentre quella inferiore rappresenta il target qubit. L'azione del gate può essere descritta dicendo che se il control qubit è nello stato $|0\rangle$, allora il target qubit viene lasciato inalterato. Se, invece, il control qubit è nello stato $|1\rangle$, allora sul target qubit viene applicato il gate $X$. In equazioni:

$$|00\rangle \to |00\rangle; \quad |01\rangle \to |01\rangle; \quad |10\rangle \to |11\rangle; \quad |11\rangle \to |10\rangle \tag{11}$$

Da qui il nome CNOT. Notiamo che è una generalizzazione dello XOR classico; infatti la sua azione può essere riassunta in $|A, B\rangle \to |A, B \oplus A\rangle$ dove $\oplus$ è l'addizione modulo due. Abbiamo utilizzato il termine *generalizzazione* e non equivalenza, in quanto, mentre lo XOR classico non è invertibile, il CNOT quantistisco lo è (come tutti gli altri gate quantistici unitari).

Questo gate è particolarmente importante; infatti, **qualsiasi gate che agisce su un numero arbitrario di qubits può essere decomposto in CNOT e gate che agiscono su un solo qubit**.

## 2.3 Circuiti quantistici

Un circuito quantistico è costituito da un insieme finito di gates quantistici applicati in **sequenza** ad un registro di qubits, inizializzato per convenienza nello stato $|00\ldots0\rangle$, con ciascun qubit nello stato $|0\rangle$. Eseguiti tutti i gate, si compie una misura proiettiva sul registro di qubit, così da far collassare il sistema in un autostato che sarà il risultato della computazione.

Un esempio di circuito quantistico è

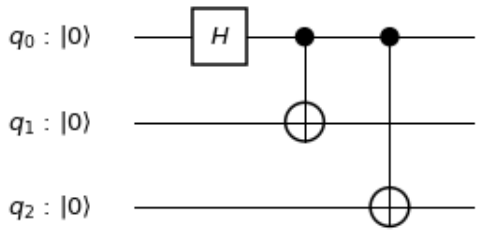



Figura 2: Circuito GHZ

Il circuito va letto da sinistra verso destra ed è importante l'ordine di applicazione dei gate, perché questi in generale **non commutano**. In questo specifico esempio, il circuito porta i qubits nella seguente sequenza di stati:

$$|000\rangle \longrightarrow \frac{1}{\sqrt{2}}(|000\rangle + |100\rangle) \longrightarrow \frac{1}{\sqrt{2}}(|000\rangle + |110\rangle) \longrightarrow \frac{1}{\sqrt{2}}(|000\rangle + |111\rangle) \tag{12}$$

Quello creato è uno stato in cui si ha entanglement a tre qubit e che è noto come stato GHZ[1]. Si potrebbe fare ora una misura sul sistema ed ottenere con uguale probabilità 000, oppure 111. In genere si affianca al registro quantistico un semplice registro classico di bits in cui si memorizza il risultato della misura.

Notiamo che, a differenza dei circuiti classici, nei circuiti quantistici **non ci sono cicli**.

## 2.4 Computazione quantistisca

L'obiettivo di qualsiasi circuito classico è quello di implementare una funzione che, presi in input un insieme di bit in un generico stato del tipo $x = 010\cdots1$, produca un output del tipo $y = 110\cdots0$, che può, in generale, richiedere un numero diverso di bit da quello di input. Cioè, un circuito calcola il valore della funzione

$$\begin{aligned} f : \{0,1\}^n &\longrightarrow \{0,1\}^m \\ x &\longmapsto f(x) = y \end{aligned} \tag{13}$$

Analogamente, quello che fa un circuito quantistico può essere schematizzato come segue:

$$U : |z, 0\rangle \to |z, f(z)\rangle \tag{14}$$

dove $|z\rangle$ è un generico stato di un sistema di $n$ qubit, mentre $|0\rangle$ è lo stato inizializzato a $|00\cdots0\rangle$ di un sistema di $m$ qubit. Questo presenta dei vantaggi rispetto alla generica funzione classica sopra specificata. Per capirlo, supponiamo di avere in input un registro di un qubit. Costruiamo un generico circuito che implementa un operatore $U$ t.c.

$$U : |0, 0\rangle \to |0, f(0)\rangle \tag{15}$$

$$U : |1, 0\rangle \to |1, f(1)\rangle \tag{16}$$

Questo è quello che potrebbe fare anche una qualsiasi funzione classica. Tuttavia, lo stato di partenza del qubit può anche essere

$$|z\rangle = \alpha|0\rangle + \beta|1\rangle \tag{17}$$

[1]È un esempio famoso che si usa per mostrare come la M.Q. è una teoria non locale

e, se applichiamo il circuito su questo stato, otteniamo

$$U|z,0\rangle = \alpha|0,f(0)\rangle + \beta|1,f(1)\rangle \tag{18}$$

Il risultato contiene l'informazione sia su $f(0)$ che su $f(1)$ ed è come se avessimo valutato $f(z)$ per due valori di $z$ simultaneamente. Si parla di **parallelismo quantistisco** intrinseco, che deriva essenzialmente dal principio di sovrapposizione e dalla linearità della meccanica quantistica. Se aggiungiamo il fatto che lo spazio di Hilbert di un sistema di $N$ qubit ha dimensione che cresce esponenzialmente con $N$, si arriva alla conclusione di poter calcolare simultaneamente una funzione su $2^N$ input diversi.

$$U \sum_{x=00\ldots0}^{11\ldots1} C_x|x,0\rangle = \sum_{x=00\ldots0}^{11\ldots1} C_x U|x,0\rangle = \sum_{x=00\ldots0}^{11\ldots1} C_x|x,f(x)\rangle \tag{19}$$

Tuttavia, notiamo che questo fatto non è immediatamente utile in quanto, per ottenere dei risultati concreti, dobbiamo compiere una misura sul sistema, che fa collassare i qubit in un unico autostato. Alla fine, quindi, abbiamo inevitabilmente accesso ad un unico valore della funzione, come nella computazione classica. *Lo scopo di un algoritmo quantistico è quello di trarre vantaggio da questo parallelismo intrinseco ed è proprio quello che fa l'algoritmo di Grover, come vedremo.*

Notiamo che il parallelismo della computazione quantistica è di *natura differente* rispetto a quello classico: classicamente risolvere parallelamente un problema significa impiegare più circuiti facendoli lavorare contemporaneamente ed assegnando a ciascuno una parte del calcolo complessivo. In questo caso, dunque, sfruttando più circuiti possiamo calcolare allo stesso tempo la funzione su diversi input, assegnando a ciascun circuito il calcolo di $f$ su un determinato input. Quantisticamente, usiamo **un solo** circuito che valuta la funzione $f$ su più valori, sfruttando la sovrapposizione.

# 3 Algoritmo di Grover

## 3.1 Setup

Il problema che vogliamo risolvere è quello della ricerca lineare.

**Formulazione 1** Sono dati un **database non strutturato**[2] di elementi ed un **oracolo**[3]. Il problema è quello di trovare quale elemento dell'insieme soddisfi l'oracolo.

Il problema può essere generalizzato a situazioni in cui l'elemento da cercare è più di uno. Invece che cercare direttamente sugli oggetti, possiamo associare loro un indice e concentrarci su questo, riformulando il problema.

**Formulazione 2** Sono dati un insieme di stringhe
$X = \{00\ldots0, 10\ldots0, \cdots\cdots, 11\ldots1\}$ ed una funzione $f(x)$, con $x \in X$, t.c. $f(x) = 1$ se $x$ è soluzione del problema di ricerca, viceversa $f(x) = 0$. Il problema è quindi quello di trovare $x$ t.c. $f(x) = 1$.

Assumiamo che la soluzione del problema sia unica.
Classicamente non esistono soluzioni efficienti al problema: quello che si deve fare è valutare la funzione $f$ su ciascun elemento dell'insieme, fin quando non si trova la soluzione. Definiamo in questo contesto *la complessità computazionale come il numero di volte che bisogna applicare la funzione f per risolvere il problema. L'obiettivo di un efficiente algoritmo di ricerca deve essere quello di minizzare il numero di chiamate dell'oracolo.* La complessità dell'algoritmo di ricerca lineare classico, se $N$ sono gli elementi dell'insieme, è, evidentemente, $O(N)$. La funzione $f$, o **oracolo**, è considerata come un **dato** del problema, una blackbox che ci viene fornita insieme al database e che è in grado di **riconoscere** la soluzione. Per enfatizzare questo aspetto, facciamo un esempio. Supponiamo che ci venga dato un numero intero $m$ e che ci venga detto che è prodotto di due numeri primi $p$ e $q$: dobbiamo determinare quali sono questi numeri. Per risolvere il problema l'algoritmo

[2]Non essendo presenti strutture al suo interno non possiamo ordinare in alcun modo i suoi elementi

[3]In informatica, un oracolo è una regola che consente di stabilire, attraverso un certo numero di operazioni o verifiche solitamente semplici e rapide, se dati enti soddisfano o meno una data proprietà o relazione.

più semplice che si può pensare è quello di scrivere tutti i numeri da 2 ad $m^{\frac{1}{2}}$ e provare la divisione di $m$ per ciascuno di questi fin quando non si trova quello che dà come resto zero. Trovato il primo numero l'altro si può ottenere dividendo $m$ per esso. Abbiamo, quindi, ricondotto il problema della fattorizzazione ad uno di ricerca lineare. In questo esempio l'oracolo deve essere una funzione in grado di stabilire se un intero è divisore o meno di $m$. È importante notare che possiamo costruirlo senza conoscere la soluzione del problema. Infine, questo non è un algoritmo efficiente; infatti, i confronti da fare aumentano esponenzialmente all'aumentare del numero di cifre di $m$.

L'algoritmo di Grover offre una soluzione più efficiente di quella classica al problema della ricerca lineare; infatti, la complessità computazionale si riduce a $O(\sqrt{N})$. Il motivo per cui questo è possibile è proprio quello discusso nell'introduzione: la sovrapposizione ed il parallelismo intrinseco della computazione quantistica.

Quantisticamente, possiamo mettere in corrispondenza gli elementi dell'insieme $X$ con gli stati di base di uno spazio di Hilbert. Se $X$ contiene $N$ elementi, la dimensione dello spazio di Hilbert deve anche essere $N$ e, dunque, i qubit da utilizzare sono $n = \lfloor \log_2 N \rfloor + 1$. L'oracolo $f$ è rappresentato da un operatore unitario $\hat{O}$ che opera sugli stati di base $|x\rangle$ la seguente trasformazione

$$|x\rangle|q\rangle \xrightarrow{\hat{0}} |x\rangle|q \oplus f(x)\rangle \tag{20}$$

dove $x$ è l'indice dell'elemento nel registro e $|q\rangle$ è lo stato di un singolo qubit, che viene scambiato ($|0\rangle \leftrightarrow |1\rangle$) se $x$ è soluzione e rimane inalterato altrimenti. A questo punto, potremmo trovare la soluzione del problema applicando $\hat{O}$ allo stato $|x\rangle|0\rangle \ \forall x$ fin quando non troviamo quello per cui lo stato del qubit ausiliario non diventa $|1\rangle$. Osserviamo che, se inizializziamo il qubit ausiliario nello stato $(|0\rangle - |1\rangle)/\sqrt{2}$, se $x$ non è soluzione non viene modificato, mentre se $x$ è soluzione, esso viene modificato in $-(|0\rangle - |1\rangle)/\sqrt{2}$. L'azione dell'oracolo può quindi essere schematizzata come

$$|x\rangle \left( \frac{|0\rangle - |1\rangle}{\sqrt{2}} \right) \xrightarrow{o} (-1)^{f(x)}|x\rangle \left( \frac{|0\rangle - |1\rangle}{\sqrt{2}} \right) \tag{21}$$

Notiamo che lo stato del qubit ausiliario non cambia, possiamo quindi ometterlo semplificando la descrizione dell'oracolo che può essere scritto come

$$|x\rangle \xrightarrow{O} (-1)^{f(x)}|x\rangle \tag{22}$$

o, equivalentemente, se chiamiamo $\omega$ la soluzione del problema

$$\hat{O} = I - 2|\omega\rangle\langle\omega| \tag{23}$$

## 3.2 Iterazione di Grover

Per eseguire l'algoritmo[4], come anticipato, abbiamo bisogno di un registro di $n$ qubit, inizializzato nello stato $|0\rangle^{\otimes n}$. Applicando Hadamard su ciascun qubit (l'operatore corrispondente è $H^{\otimes n}$), possiamo ottenere lo stato

$$|s\rangle = \frac{1}{\sqrt{N}} \left( \sum_{x=0}^{N-1} |x\rangle \right) \tag{24}$$

questo è, allo stesso tempo e con la stessa probabilità, sovrapposizione di tutti gli stati di base su cui stiamo effettuando la ricerca. Se eseguissimo ora una misura sul sistema, la probabilità di trovare la risposta esatta sarebbe

$$|\langle\omega|s\rangle|^2 = \frac{1}{N} \tag{25}$$

L'idea alla base dell'algoritmo di Grover è di compiere sullo stato una serie di operazioni in maniera tale da aumentare il coefficiente del termine $|\omega\rangle$ e diminuire tutti gli altri, cosicché, quando si esegue la misura, il sistema collassa con alta probabilità nello stato corrispondente alla soluzione del problema.

Costruiamo un'iterazione di Grover combinando l'azione dell'oracolo con l'operatore

$$\hat{U} = 2|s\rangle\langle s| - I \tag{26}$$

Sia l'oracolo che $\hat{U}$ sono delle riflessioni, la prima rispetto all'iperpiano ortogonale a $|\omega\rangle$ e contenente tutte le altre direzioni, la seconda rispetto all'iperpiano ortogonale a $|s\rangle$ e contenente tutte le altre direzioni; infatti, le componenti lungo queste ultime non vengono modificate. Un'iterazione di Grover è quindi data dall'applicazione dell'operatore

[4]Ricordiamo che siamo nel caso in cui sappiamo che la soluzione è unica.

$$\hat{G} = \hat{U}\hat{O} \tag{27}$$

La composizione di due riflessioni ci dà una rotazione, che avviene nel piano generato da $|\omega\rangle$ e $|s\rangle$. Rappresentiamo come viene modificato lo stato dopo la prima iterazione

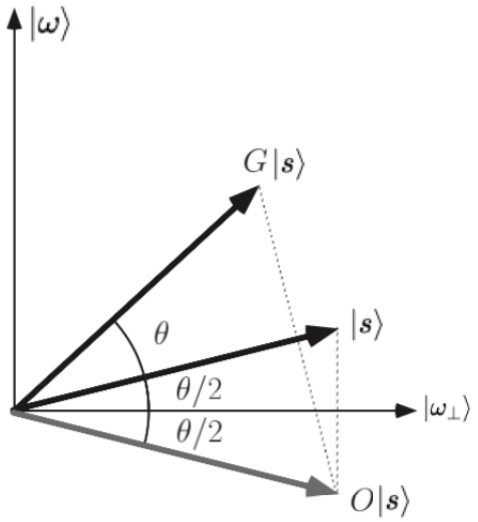



Figura 3: Azione della prima iterazione di Grover

Lo stato di partenza, $|s\rangle$, ha aumentato la sua componente lungo la soluzione $|\omega\rangle$, mentre le componenti lungo tutte le altre direzioni sono diminuite. Poiché

$$|\omega'_\perp\rangle = |s\rangle - \frac{1}{\sqrt{N}}|\omega\rangle \tag{28}$$

che, normalizzato, diventa

$$|\omega_\perp\rangle = \sqrt{\frac{N}{N-1}}\left\{|s\rangle - \frac{1}{\sqrt{N}}|\omega\rangle\right\}, \tag{29}$$

abbiamo che

$$\sin\frac{\theta}{2} = \frac{1}{\sqrt{N}} \tag{30}$$

Vogliamo ora dimostrare che, ad ogni iterazione, lo stato viene ruotato di $\theta$ verso la soluzione $|\omega\rangle$. Dobbiamo, quindi, dimostrare che la matrice rappresentativa di $\hat{G}$ è ovunque nulla, tranne che nel sottospazio generato da $|\omega\rangle$ e $|\omega_\perp\rangle$ in cui è

$$\hat{G}_{sottospazio} \doteq \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \text{ [5]} \tag{31}$$

Per farlo, basta ricordare la definizione di $\hat{G}$

$$\begin{aligned} \hat{G} = \hat{U}_s\hat{O} &= (-I + 2|s\rangle\langle s|)(I - 2|\omega\rangle\langle\omega|) \\ &= -I + 2|\omega\rangle\langle\omega| + 2|s\rangle\langle s| - 4\sin\frac{\theta}{2}|s\rangle\langle\omega| \end{aligned} \tag{32}$$

A questo punto per calcolare gli elementi di matrice dell'operatore, basta prenderne il prodotto "bra-ket" tra i diversi stati di base. Calcoliamo esplicitamente il primo elemento di matrice nel sottospazio considerato

$$\begin{aligned} \langle\omega|\hat{G}|\omega\rangle &= -1 + 2 + 2|\langle s|\omega\rangle|^2 - 4\sin\frac{\theta}{2}\langle s|\omega\rangle \\ &= 1 - 2\sin^2\frac{\theta}{2} = \cos\theta \end{aligned} \tag{33}$$

Consideriamo, invece, un elemento di matrice al di fuori del sottospazio considerato. Prendiamo $|\alpha\rangle$ t.c. $\langle\alpha|\omega\rangle = \langle\alpha|\omega_\perp\rangle = 0$[6], allora

$$\begin{aligned} \langle\omega|\hat{G}|\alpha\rangle &= 2\langle\omega|s\rangle\langle s|\alpha\rangle \\ &= 2\langle\omega|s\rangle\left(\langle\omega_\perp| + \frac{1}{\sqrt{N}}\langle\omega|\right)|\alpha\rangle = 0 \end{aligned} \tag{34}$$

Calcolando tutti gli altri elementi di matrice, si trova il risultato (30) e quindi che, ad ogni iterazione dell'algoritmo di Grover, lo stato del sistema viene ruotato di $\theta$ verso la soluzione $|\omega\rangle$. Possiamo calcolare facilmente il numero di iterazioni necessarie per portare lo stato dei qubit vicino alla soluzione: dopo $T$ iterazioni, lo stato è ruotato rispetto a $|\omega_\perp\rangle$ di $\frac{\theta}{2} + T\theta$; per massimizzare la probabilità di trovare lo stato $|\omega\rangle$ quando alla fine dell'algoritmo facciamo la misura sui qubit, dobbiamo ripetere l'algoritmo finché l'angolo di cui abbiamo ruotato non è di circa $\frac{\pi}{2}$

[5] Notiamo che i segni degli elementi diagonali sono opposti rispetto alla tipica matrice di rotazione: ciò è giustificato dal fatto che stiamo ruotando in senso antiorario.

[6] $|\alpha\rangle$ non può coincidere con uno stato dell'insieme X, altrimenti $\langle\alpha|\omega_\perp\rangle = \frac{1}{\sqrt{N}}$

$$\frac{\theta}{2} + T\theta \simeq \frac{\pi}{2} \Rightarrow 2T + 1 \simeq \frac{\pi}{\theta}\,. \tag{35}$$

Nel limite di $N$ grande (supponendo quindi di avere tanti elementi), la (29) diventa

$$\frac{\theta}{2} = \frac{1}{\sqrt{N}} + O\left(\frac{1}{N}\right) \tag{36}$$

e sostituendo nella (34) abbiamo che, sempre nel limite $N \gg 1$, il numero ottimale di iterazioni è

$$T = \left\lceil \frac{\pi}{4}\sqrt{N} \right\rceil\,. \tag{37}$$

Dopo aver compiuto queste iterazioni, se si esegue la misura, la probabilità di ottenere il risultato corretto è

$$Prob(\omega) = \sin^2\left(\frac{\theta}{2} + T\theta\right) = 1 - O\left(\frac{1}{N}\right) \tag{38}$$

In conclusione, con l'algoritmo di Grover, per ottenere la soluzione del problema di ricerca lineare dobbiamo applicare l'oracolo circa $\frac{\pi}{4}\sqrt{N}$ volte, dunque abbiamo uno speedup quadratico rispetto all'algoritmo classico. La complessità di questo algoritmo è $O(\sqrt{N})$.
*Notiamo che, eseguendo più iterazioni di quelle ottimali, ci allontaniamo dalla soluzione.*

**Riepilogo dell'algoritmo**

| | |
|---|---|
| 1. $\|0\rangle^{\otimes n}\|0\rangle$ | stato iniziale |
| 2. $\to \frac{1}{\sqrt{N}} \sum_{x=1}^{N} \|x\rangle \left[\frac{\|0\rangle - \|1\rangle}{\sqrt{2}}\right]$ | applichiamo $H^{\otimes n}$ ai primi $n$ qubit, e HX all'ultimo qubit |
| 3. $\to \hat{G}^T \left\{ \frac{1}{\sqrt{N}} \sum_{x=1}^{N} \|x\rangle \left[\frac{\|0\rangle - \|1\rangle}{\sqrt{2}}\right] \right\}$ $\approx \|\omega\rangle \left[\frac{\|0\rangle - \|1\rangle}{\sqrt{2}}\right]$ | applichiamo l'operatore di Grover $T \simeq \pi \frac{\sqrt{N}}{4}$ volte |
| 4. $\to \omega$ | misura sui primi $n$ qubit |

(39)

## 3.3 Inversione rispetto alla media

Vi è un modo semplice di visualizzare l'azione dell'operatore $\hat{U}$. Se espandiamo un generico stato $|\psi\rangle$ nella base computazionale

$$|\psi\rangle = \sum_x a_x |x\rangle \tag{40}$$

allora il suo prodotto scalare con $|s\rangle = \frac{1}{\sqrt{N}} \sum_x |x\rangle$ è

$$\langle s|\psi\rangle = \frac{1}{\sqrt{N}} \sum_x a_x = \sqrt{N}\langle a\rangle \tag{41}$$

dove

$$\langle a\rangle = \frac{1}{N} \sum_x a_x \tag{42}$$

è l'ampiezza media dei coefficienti dello stato. Applicando l'operatore $\hat{U} = 2|s\rangle\langle s| - 1$ allo stato $|\psi\rangle$, otteniamo

$$\hat{U}|\psi\rangle = \sum_x \left(2\langle a\rangle - a_x\right)|x\rangle \tag{43}$$

Per comprendere come viene modificato lo stato, ci riferiamo alla seguente figura, che mostra come vengono modificati i coefficienti dopo la prima iterazione.

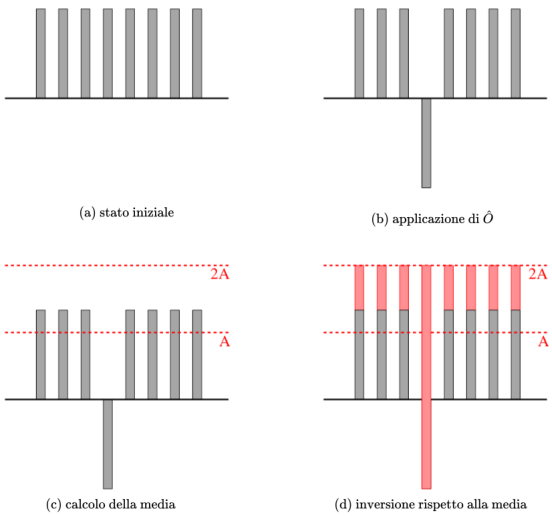



Figura 4: Rappresentazione di un'iterazione di Grover

L'immagine (a) mostra lo stato iniziale, sovrapposizione uniforme di tutti gli stati di base $|x\rangle$, su cui stiamo effettuando la ricerca. Quelle rappresentate sono proprio le ampiezze di probabilità, che in questo caso sono reali. In (b) abbiamo l'applicazione dell'oracolo, che cambia di segno il coefficiente dello stato soluzione $|\omega\rangle$: notiamo che, se potessimo accorgerci del cambiamento di fase di uno dei coefficienti, potremmo concludere immediatamente l'algoritmo; tuttavia, non si può estrarre questa informazione dal sistema con una misura. Nell'immagine (c), calcoliamo la media dei coefficienti e notiamo che le ampiezze relative agli stati che non sono soluzione sono al di sopra di essa; il viceversa vale per $|\omega\rangle$. In (d), si vede come vengono modificati i coefficienti dopo l'applicazone di $\hat{U}$: quelle in rosso sono le nuove ampiezze. Notiamo che tutte le ampiezze che stavano al di sopra della media ora stanno al di sotto e viceversa: in questo modo l'ampiezza relativa allo stato soluzione aumenta.

Ricordiamo che questo è soltanto un altro modo di visualizzare un'iterazione di Grover, ed è del tutto equivalente alla rappresentazione fatta in Figura (4).

## 3.4 Generalizzazione al caso di M soluzioni

Vogliamo ora estendere l'algoritmo di Grover al caso in cui nel database vi siano $M > 1$ elementi che soddisfano il criterio di ricerca. *Vedremo che l'unica modifica da apportare all'algoritmo è il numero di volte ottimale dell'applicazione dell'oracolo.*

Partiamo sempre dallo stato sovrapposizione uniforme, $|s\rangle = \frac{1}{\sqrt{N}} \left( \sum_{x=0}^{N-1} |x\rangle \right)$; tuttavia, invece che vederlo come $|s\rangle = \frac{1}{\sqrt{N}}|\omega\rangle + |\omega_\perp\rangle$ dove $|\omega\rangle$ è uno stato di base e cioè l'unica soluzione, lo decomponiamo in

$$|\tilde{\omega}\rangle = \frac{1}{\sqrt{M}} \left( \sum_{i=1}^{M} |\omega_i\rangle \right) \tag{44}$$

e

$$|\tilde{\omega}_\perp\rangle = \frac{1}{\sqrt{N-M}} \left( \sum_{x \neq \omega_i} |x\rangle \right) , \tag{45}$$

cioè nello stato sovrapposizione uniforme di tutte le soluzioni, $|\tilde{\omega}\rangle$, e nello stato sovrapposizione uniforme di tutte le *non* soluzioni. È semplice vedere che possiamo riscrivere lo stato iniziale $|s\rangle$ come

$$|s\rangle = \sqrt{\frac{N-M}{N}}|\tilde{\omega}_\perp\rangle + \sqrt{\frac{M}{N}}|\tilde{\omega}\rangle \tag{46}$$

Se confrontiamo con l'espressione (28), notiamo che questa è del tutto analoga. Possiamo, dunque, procedere esattamente come nel caso di soluzione unica, soltanto che, ora, la rotazione dovuta ad ogni iterazione di Grover avviene nel piano $\mathrm{Span}\{|\tilde{\omega}\rangle, |\tilde{\omega}_\perp\rangle\}$, l'angolo iniziale è $\sin(\frac{\theta}{2}) = \sqrt{\frac{M}{N}}$ e, dunque, ciò che cambia è il numero di iterazioni ottimali che diventa

$$T = \left\lceil \frac{\pi}{4}\sqrt{\frac{N}{M}} \right\rceil . \tag{47}$$

Osserviamo che, alla fine dell'algoritmo, per ottenere il risultato, dobbiamo effettuare una misura sul sitema di qubit. Questo ci permetterà di ottenere soltanto una delle soluzioni del problema, ciascuna con la stessa probabilità. Inoltre, il numero di iterazioni ottimali è funzione del numero di soluzioni $M$, che bisogna conoscere a priori.

# 4 Implementazione numerica dell'algoritmo di Grover

Implementiamo l'algoritmo di Grover numericamente utilizzando *qiskit*, una libreria di python che consente di simulare un computer quantistico.
Con la ricerca lineare possiamo trovare le soluzioni di un qualsiasi problema decisionale NP. Un problema decisionale è rappresentato da una funzione del tipo

$$f : \{0,1\}^n \to \{0,1\} \tag{48}$$

e risolverlo vuol dire trovare tutte le stringhe di $n$ bit sulle quali l'applicazione della funzione $f$ produce come output 1, cioè 'Vero'. I problemi della classe NP sono caratterizzati dal fatto che è possibile verificare, in un tempo polinomiale nella lunghezza della stringa di input ($n$), se un input è soluzione o meno, ma non si può costruire un circuito che in un tempo polinomiale in $n$ trovi tutte le soluzioni del problema. In altri termini per un problema della classe NP è semplice verificare se un input è soluzione del problema, ma è difficile individuare in modo efficiente tutte le soluzioni del problema. Il tempo richiesto per risolvere un problema di tipo NP dipende dall'algoritmo che si utilizza e, in genere, è *esponenziale* nella lunghezza dell'input. Qualsiasi problema NP può essere risolto con un algoritmo di ricerca lineare: il modo più semplice per trovare tutte le soluzioni è quello di verificare se ogni possibile stringa risolve il problema. Questo metodo risolutivo non è efficiente[7], infatti un algoritmo classico di ricerca lineare avrebbe complessità $O(2^n)$. Con l'algoritmo di Grover possiamo ridurre la complessità quadraticamente e cioè avremo $O(2^{\frac{n}{2}})$. L'algoritmo, dunque, continua ad essere esponenziale.

## 4.1 Introduzione al 3-SAT

Il **3-SAT** Problem (o Satisfiability Problem) è un problema decisionale NP-completo. Ci si chiede se, date $N$ variabili booleane $x_1, x_2, \cdots, x_N \in 0, 1$ e quindi delle stringhe $x_1 x_2 \cdots x_N$, ne esista una che rende vera una formula

[7]Per efficiente si intende un algoritmo in grado di risolvere il problema in un tempo polinomiale nella grandezza degli input. Sottolineaimo che in generale non è noto se un problema della classe NP ammette un algoritmo in grado di risolvere il problema efficientemente ($P \stackrel{?}{=} NP$).

booleana. Se la risposta è affermativa la formula si dice soddisfacibile. La formula booleana è, in generale, composta combinando le variabili $x_1, x_2, \cdots, x_N$ utilizzando AND, OR e NOT e può sempre essere ricondotta al caso in cui è espressa in forma normale congiuntiva (CNF), ovvero come un AND di clausole, ognuna formata da un OR di letterali. Per letterali intendiamo le variabili booleane o la loro negazione. Il nome 3-SAT deriva dal fatto di costruire funzioni utilizzando massimo 3 letterali per clausola.
Non risolveremo il problema del 3-SAT ma ci proponiamo di trovare la soluzione della seguente funzione booleana *sapendo che è unica.*

$$f(x_1, x_2, x_3) = \neg x_1 \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee x_2) \tag{49}$$

Il numero totale degli elementi sui quali effettuare la ricerca è, in questo esempio, il numero di tutte le possibili stringhe ottenibili a partire da tre variabili booleane, e cioè $N = 8$. Data l'unicità della soluzione possiamo utilizzare la formula (36) per determinare il numero ottimale di iterazioni di Grover da compiere; tuttavia, questa è stata ricavata nel limite $N \gg 1$. Nell'esempio considerato $N = 8$, conviene allora utilizzare la relazione completa

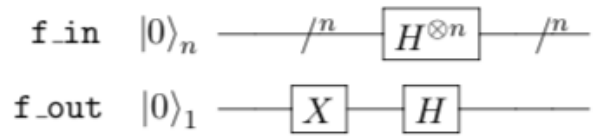
$$T = \left\lceil \frac{\pi}{4 \arcsin(\frac{1}{\sqrt{N}})} - \frac{1}{2} \right\rceil \tag{50}$$

che, per $N = 8$, fornisce $T = 2$.

## 4.2 Costruzione del circuito

Per applicare l'algoritmo di Grover abbiamo bisogno di tre subroutine: una per creare lo stato iniziale, una per costruire l'oracolo $\hat{O}$ e una per eseguire l'inversione rispetto alla media. Inoltre poiché il database è costituito da 8 elementi abbiamo bisogno di un registro di 3 qubit per descriverlo, mentre per scrivere il risultato è necessario un solo qubit. Chiamiamo rispettivamente **f_in** ed **f_out** questi registri.

**Stato iniziale:** Per costruire lo stato iniziale applichiamo il seguente circuito




In questo modo creiamo lo stato $|s\rangle$ per gli n qubit, mentre inizializzamo il qubit di output nello stato $(|0\rangle - |1\rangle)/\sqrt{2}$ come richiesto dall'algoritmo di Grover.

**Oracolo:** Ci serve costruire un operatore che sappia riconoscere la soluzione del problema, questo può essere fatto in diversi modi. Per semplicità utilizzeremo un registro di tre qubit ausiliari (uno per ogni clausola), costruendo un circuito per ogni clausola che scambia il corrispondente qubit ausiliario se la clausola è soddisfatta. I qubit ausiliari saranno inizializzati nell'autostato $|0\rangle$.
Utilizzeremo i teoremi di De Morgan, che possono essere formulati come segue

$$\begin{aligned} \neg(P \wedge Q) &= (\neg P) \vee (\neg Q) \\ \neg(P \vee Q) &= (\neg P) \wedge (\neg Q) \end{aligned} \tag{51}$$

dove $P$ e $Q$ sono proposizioni logiche. Applicati, ad esempio, alla seconda clausola producono

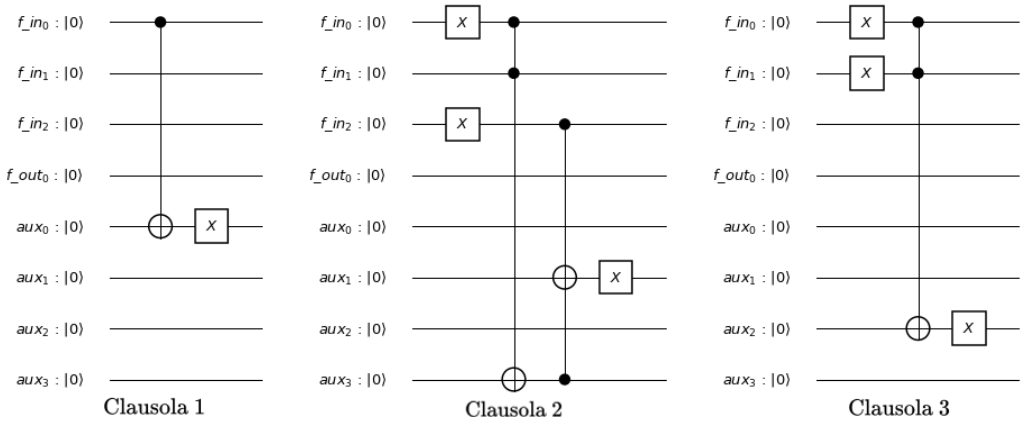
$$\neg(x_1 \vee \neg x_2 \vee x_3) = \neg x_1 \wedge x_2 \wedge \neg x_3 \tag{52}$$

quindi possiamo scrivere che

$$(x_1 \vee \neg x_2 \vee x_3) = \neg(\neg x_1 \wedge x_2 \wedge \neg x_3) \tag{53}$$

Questo semplifica il problema; infatti, intuitivamente: prendendo sempre come esempio la seconda clausola, questa sarà sempre vera a meno che $x_1 = 0, x_2 = 1, x_3 = 0$. Per verificare se è soddisfatta basta verificare che la stringa in ingresso non sia proprio 010, che è quello che ci dicono i teoremi di De Morgan ed è quello che faremo.

Riportiamo i circuiti che verificano la veridicità della clausole.




I gate utilizzati sono X e CNOT, presentati nell'introduzione, ed il CC-NOT che è identico al CNOT ma con due qubit di controllo. Notiamo che, per la seconda clausola, sarebbe sufficiente un triplo-CNOT; questo, tuttavia, non è presente in qiskit. Allora lo implementiamo come mostrato in figura, utilizzando un qubit ausiliario in più. Possiamo facilmente verificare che il secondo qubit ausiliario viene scambiato soltanto se tutti i qubit in ingresso sono nello stato 1. In accordo con i teoremi di De Morgan, come ultimo passo nella verifica di ciascuna clausola, applichiamo $X$ al qubit ausiliario corrispondente, che è stato reso vero se e solo se la clausola non è soddisfatta.
Rimane infine da verificare che tutte le clausole siano soddisfatte, e questo possiamo farlo con un circuito analogo a quello usato nella clausola 2.

Sottolineiamo che abbiamo implementato l'oracolo **senza conoscere la soluzione** del problema. Riportiamo il circuito completo dell'oracolo

Come si vede, abbiamo infine riapplicato le stesse operazioni per resettare lo stato iniziale dei qubit, fase necessaria prima dei successivi passaggi.

**Inversione rispetto alla media ($\hat{U}$):** Per implementare l'operatore $\hat{U}$ conviene riscriverlo come segue

$$\begin{aligned} \hat{U} = 2|s\rangle\langle s| - I &= 2H^{\otimes n}|0\rangle\langle 0|H^{\otimes n} - I \\ &= H^{\otimes n}\big(2|0\rangle\langle 0| - I\big)H^{\otimes n} \end{aligned} \tag{54}$$

se applichiamo l'operatore $2|0\rangle\langle 0| - I$ ad un generico stato

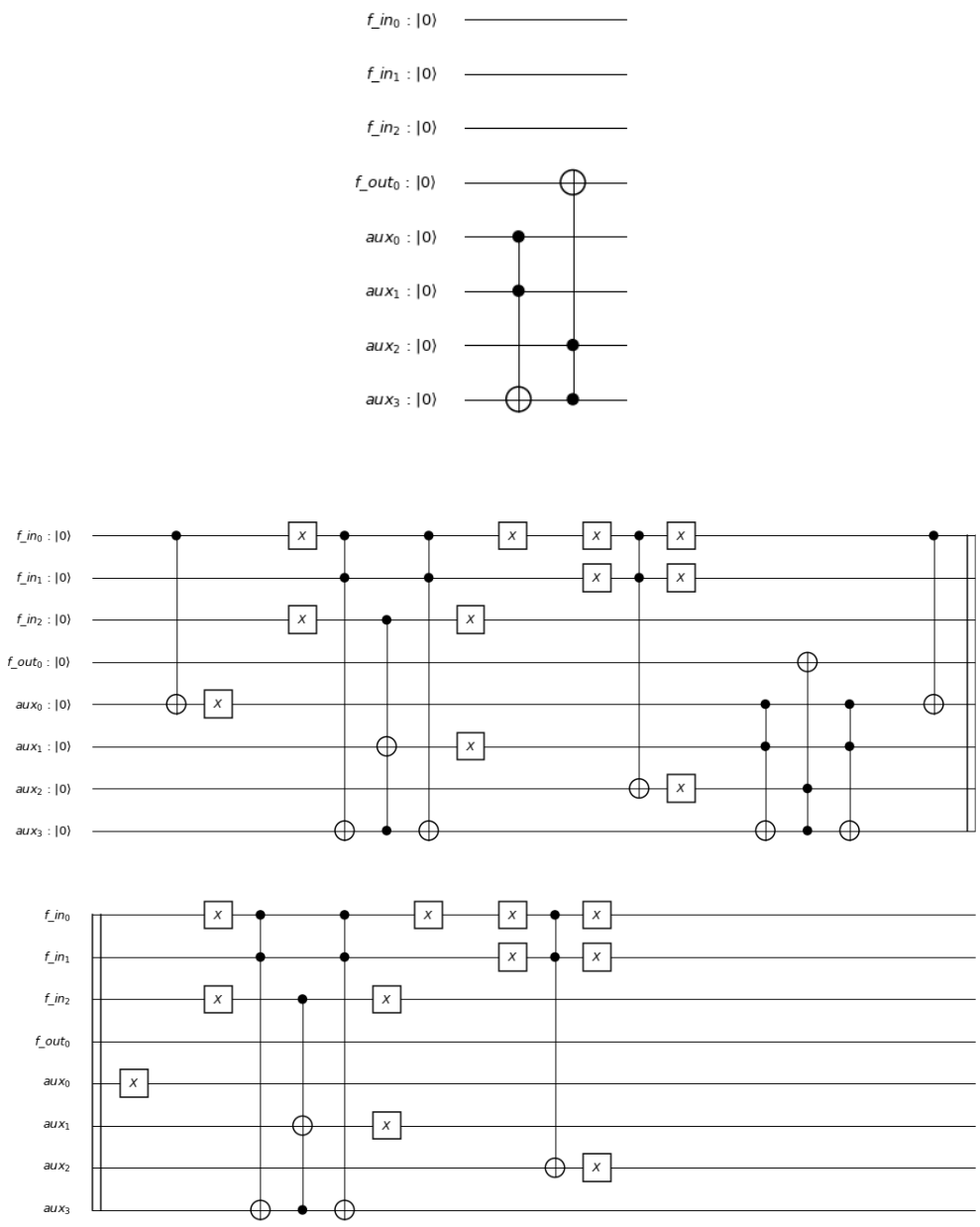



Figura 5: Circuito che costituisce l'oracolo

$$\begin{aligned}\big\{2|0\rangle\langle 0| - I\big\}|\psi\rangle &= \big\{2|0\rangle\langle 0| - I\big\}\sum_{x=0}^{N-1} c_x|x\rangle \\ &= -\left(\sum_{x=1}^{N-1} c_x|x\rangle - c_0|0\rangle\right)\end{aligned} \tag{55}$$

Trascurando il fattore di fase globale che è ininfluente, vediamo che questo operatore modifica un generico stato cambiando di segno la sua componente lungo $|0\rangle$. Possiamo allora implementare l'operatore di inversione rispetto alla media con il seguente circuito

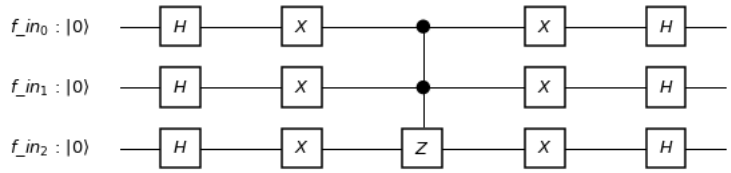



Figura 6: Circuito che esegue l'inversione rispetto alla media

Rimane soltanto da implementare il gate "CCZ", che deve applicare Z al target qubit se i due control qubit sono in $|1\rangle$. Possiamo farlo usando un CCNOT e due Hadamard

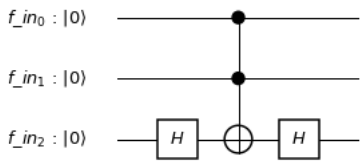



Figura 7: Circuito CCZ

Infatti, se il CCNOT non agisce sul target qubit abbiamo $H^2 = I$, mentre se agisce applica sul target $X$ e si ha $HXH = Z$.

Utilizzando queste subroutine, possiamo eseguire l'algoritmo di Grover, ricordando che il numero di iterazioni ottimale è 2. Mostriamo come vengono modificate le ampiezze di probabilità nell'esecuzione

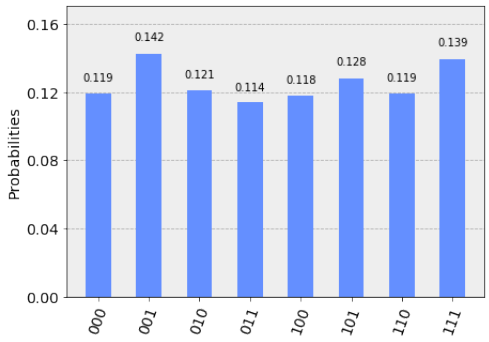



Figura 8: Probabilità iniziali per i diversi stati del registro.

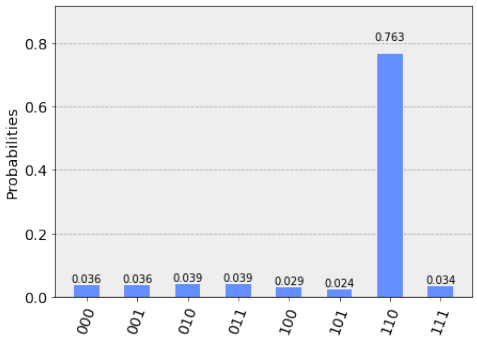



Figura 9: Probabilità dei diversi stati del registro dopo la prima iterazione.

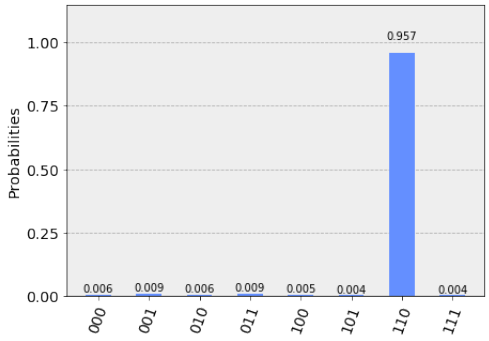



Figura 10: Probabilità dei diversi stati del registro dopo la seconda iterazione.

Se facessimo la misura subito dopo la seconda iterazione avremmo il $95,7\%$ di probabilità di far collassare lo stato dei qubit nello stato $|011\rangle$, che possiamo verificare essere l'unica soluzione della funzione (48).
**Abbiamo così risolto il problema di ricerca lineare in un database di $N = 8$ elementi con soltanto due applicazioni dell'oracolo. Classicamente, dovremmo, in media, applicarlo quattro volte.**

Se compissimo un'iterazione in più la probabilità di trovare la soluzione esatta diminuirebbe

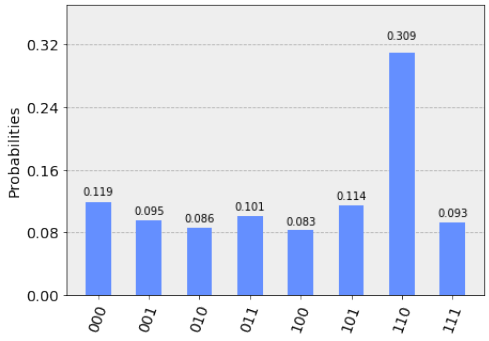



Figura 11: Probabilità dei diversi stati del registro dopo la terza iterazione.

# 5 Appendici

## 5.1 Codice qiskit

Riportiamo il codice qiskit utilizzato per fare la simulazione

```
import numpy as np
import matplotlib.pyplot as plt

# importo Qiskit
from qiskit import BasicAer, IBMQ
from qiskit import QuantumCircuit, ClassicalRegister, QuantumRegister
from qiskit import execute, compile
from qiskit.tools.visualization import plot_histogram
```

```
def oracle(circuit, f_in, f_out, aux, n):
    #clausola 1
    circuit.cx(f_in[0], aux[0])
    circuit.x(aux[0])

    #clausola 2
    circuit.x(f_in[0])
    circuit.x(f_in[2])
    circuit.ccx(f_in[0], f_in[1], aux[3])
    circuit.ccx(f_in[2], aux[3], aux[1])
    circuit.ccx(f_in[0], f_in[1], aux[3])
    circuit.x(f_in[0])
    circuit.x(f_in[2])
    circuit.x(aux[1])

    #clausola 3
    circuit.x(f_in[0])
    circuit.x(f_in[1])
    circuit.ccx(f_in[0], f_in[1], aux[2])
    circuit.x(aux[2])
    circuit.x(f_in[0])
    circuit.x(f_in[1])

    #controllo che tutte le clausole sono soddisfatte
    circuit.ccx(aux[0], aux[1], aux[3])
    circuit.ccx(aux[2], aux[3], f_out[0])
    circuit.ccx(aux[0], aux[1], aux[3])

    #ripristino lo stato iniziale dei qubit riapplicando lo stesso circ
    circuit.cx(f_in[0], aux[0])
    circuit.x(aux[0])

    circuit.x(f_in[0])
    circuit.x(f_in[2])
    circuit.ccx(f_in[0], f_in[1], aux[3])
    circuit.ccx(f_in[2], aux[3], aux[1])
    circuit.ccx(f_in[0], f_in[1], aux[3])
    circuit.x(f_in[0])
```

```
    circuit.x(f_in[2])
    circuit.x(aux[1])

    circuit.x(f_in[0])
    circuit.x(f_in[1])
    circuit.ccx(f_in[0], f_in[1], aux[2])
    circuit.x(aux[2])
    circuit.x(f_in[0])
    circuit.x(f_in[1])
    circuit.barrier()

def n_controlled_Z(circuit, controls, target):
        circuit.h(target)
        circuit.ccx(controls[0], controls[1], target)
        circuit.h(target)

def inversion_about_average(circuit, f_in, n):
    # Hadamards ovunque
    for j in range(n):
        circuit.h(f_in[j])
    # scambio il segno di |000>
    for j in range(n):
        circuit.x(f_in[j])
    n_controlled_Z(circuit, [f_in[j] for j in range(n-1)], f_in[n-1])
    for j in range(n):
        circuit.x(f_in[j])
    # Hadamards ovunque
    for j in range(n):
        circuit.h(f_in[j])

def input_state(circuit, f_in, f_out, n):
    for j in range(n):
        circuit.h(f_in[j]) #stato identita' iniziale

    circuit.x(f_out)
    circuit.h(f_out)

n = 3 #numero di variabili
```

```
f_in = QuantumRegister(n)
f_out = QuantumRegister(1)
aux = QuantumRegister(4)

#Definiamo un registro classico per memorizzare il risultato della misu
ans = ClassicalRegister(n)

grover = QuantumCircuit()
grover.add_register(f_in)
grover.add_register(f_out)
grover.add_register(aux)
grover.add_register(ans)

input_state(grover, f_in, f_out, n)

for i in range(2):
    oracle(grover, f_in, f_out, aux, n)
    inversion_about_average(grover, f_in, n)

for j in range(n):
    grover.measure(f_in[j], ans[j])

# eseguo il circuito 1000 volte e faccio una statistica dei risultati
backend = BasicAer.get_backend('qasm_simulator')
job = execute([grover], backend=backend, shots=1000)
result = job.result()

# rappresento i risultati con un istogramma
counts = result.get_counts(grover)
plot_histogram(counts)
```

# Riferimenti bibliografici

[1] Quantum Computation and Quantum Information by Isaac Chuang and Michael Nielsen

[2] An Introduction to Quantum Computing, Without the Physics; Giacomo Nannicini; IBM T.J. Watson, Yorktown Heights, NY