



Università degli Studi dell'Aquila

Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica

Corso di laurea in Informatica

Algoritmi per la computazione quantistica

Relatori

Prof. Fabrizio Rossi
Prof. Leonardo Guidoni

Candidato

Leonardo Serilli

Anno Accademico 2019/2020

A tutti gli ignoranti, inclini alla curiosità.

Introduzione

John von Neumann nel 1945 sviluppò un semplice modello teorico che riuniva i componenti necessari ad un computer per essere in grado di eseguire tutte le operazioni che una Macchina di Turing fosse in grado di eseguire, guidato dalla tesi detta di Churchill-Turing la quale asserisce che un qualsiasi modello di calcolo realizzabile possa essere simulato su una macchina di Turing. L'architettura di von Neumann venne realizzata solo con l'invenzione del Transistor nel 1947, un componente elettronico utilizzato all'interno di un circuito elettrico come amplificatore o come interruttore. Da lì venne previsto, dalla legge di Moore, che il potere computazionale dei computer, misurato in numero i transistor, raddoppiasse ogni 18 mesi. Dopo 50 anni, la legge di Moore si avvicina al termine, i transistor attualmente sono grandi 14nm che equivalgono a un ottavo del diametro del virus dell'HIV, il problema risiede nel fatto che se progettassimo transistor della grandezza di qualche atomo, sarebbero soggetto alle leggi della meccanica quantistica, e non sarebbero più in grado di interrompere il flusso di elettroni, poiché questi, per un effetto detto 'tunnelling quantistico', passerebbero da una zona all'altra del componente con effetti imprevedibili.

Una possibile soluzione al vicolo cieco posto dalla legge di Moore è stata introdotta da un nuovo paradigma di computazione: la computazione quantistica, basata sull'utilizzo della meccanica quantistica, anzichè della fisica classica, per la computazione. È stato dimostrato che un computer classico può essere usato per la simulazione di uno quantistico, ma sembrerebbe impossibile che vi riesca in modo efficiente, per via degli effetti quantistici che sfruttano, come l'entanglement, la sovrapposizione di stati ed altri, i quali ci permettono di eseguire algoritmi con un vantaggio nella velocità computazionale talmente significativo, a volte esponenziale, che alcuni ricercatori sono convinti che non saremo mai in grado di chiudere il divario creato tra il potere di computer quantistici e classici.

La maggior parte degli algoritmi eseguiti su un hardware quantistico, tra cui quelli trattati in questa tesi, seppur incredibilmente potenti da un punto di vista teorico, sono progettati per computer quantistici futuri, i quali avranno sia un elevato numeri di qubit che un basso rumore nell'hardware, al momento la Google possiede un processore quantistico chiamato Sycamore di soli 53 qubit mentre l'IBM Quantum System One ne ha 64, di conseguenza, un importante area di ricerca è quella che si occupa di progettare algoritmi efficienti già da adesso, e il campo più promettente è quello della simulazione quantistica. Un ovvio problema che incontrano i computer classici nella simulazione di sistemi quantistici è la grande quantità di memoria richiesta per immagazzinare lo stato quantistico di un grande sistema fisico, poiché i parametri crescono esponenzialmente con il sistema, invece i computer quantistici, poiché incorporano nativamente le proprietà quantistiche di tali sistemi, non hanno bisogno di una quantità esponenziale di memoria per memorizzare un esponenziale quantità di informazione. Un esempio di tali algoritmi di simulazione è il Variational Quantum Eigensolver, algoritmo usato nella simulazione di molecole per la ricerca della minima energia dello stato fondamentale (ground state energy). Il VQE, che se da una parte è molto interessante per chimici e fisici, dall'altra trova applicazione in svariati campi, infatti la simulazione di sistemi quantistici, problema computazionalmente intrattabile per i processori classici, permette di studiare le proprietà di certe molecole o materiali, e questo trova impiego in campo parafarmaceutico per il design di nuovi farmaci, nel campo dell'energia per il design di batterie, e anche nel design di nuovi materiali.

Richard Feynman con la sua citazione: "**Nature isn't classical, dammit, and if you want to make simulation of nature, you'd better make it quantum mechanical, and by golly it's a wonderful problem, because it doesn't look so easy**", predisse lo sviluppo di una nuova tecnologia che intrecciisse i ruoli di svariate discipline scientifiche: fisiche, chimiche, informatiche

e matematiche al fine di sfruttare al meglio il potere computazionale offerto dalle leggi che governano la natura, per scoprire nuovi modi di processare l'informazione e di comunicare, di creare quindi dispositivi con possibilità che vanno ben oltre quelle degli attuali sistemi di computazione e comunicazione, e infine di far emergere nuovi aspetti della natura stessa.

In questa tesi, partendo da un'introduzione ai concetti della computazione quantistica [cap.1], parlerò dei problemi che i computer quantistici sono in grado di affrontare [cap.2] e di alcuni degli algoritmi ideati per risolverli [cap.3], in particolare mi soffermerò sull'algoritmo VQE per la simulazione quantistica di molecole, e delle differenti implementazioni dei suoi componenti [cap.4], con un esempio di esse realizzati in python [cap.5] tramite la libreria python dell'IBM chiamata Qiskit [cap.1.0]

Indice

1. Introduzione alla computazione quantistica	6
1.0 Il modulo IBM Qiskit	6
1.1 I qubit	6
1.1.1 Regola di Born	7
1.1.2 Normalizzazione	7
1.1.3 Sfera di Bloch	8
1.2 Computazione quantistica	8
1.2.1 Quantum gates	8
1.2.2 Gate per singoli qubit	9
1.2.3 Gate per singoli qubit su statevector multipli	10
1.2.4 Gate per qubit multipli	11
2. Portata della computazione quantistica	13
2.1 Computazione classica su computer quantistici	13
2.2 Le classi P, NP, PSACE e BPQ	14
3. Algoritmi quantistici	16
3.1 Classi di algoritmi quantistici	16
3.2 Algoritmi basati sulla trasformata quantistica di Fourier	16
3.2.1 Trasformata quantistica di Fourier (QFT)	16
3.2.2 stima della fase quantistica (QPE)	18
3.2.3 Algoritmo di Shor	19
3.3 Algoritmi quantistici di ricerca	21
3.4 Algoritmi per la simulazione quantistica	22
4. Variational quantum eigensolver (VQE)	23
4.1 Introduzione al VQE: problema del taglio massimo (Max-Cut)	23
4.1.1 Analogia con il problema dalla minima energia dello stato fondamentale	24
4.2 VQE nella simulazione di molecole	25
4.3 Algoritmo Variational Quantum Eigensolver	25
4.3.1 Forme variazionali	26
4.3.2 Ottimizzazione	27
5. VQE: diversi approcci verso una molecola di LiH in Qiskit	28
5.1 Simulazioni senza rumore	30
5.2 Simulazioni in presenza di rumore	33

1 Introduzione alla computazione quantistica

fonti bibliografiche: [1], [2].

1.0 Il modulo IBM Qiskit

I vari esempi di codice all'interno del documento sono realizzati tramite il **framework python open source** per la computazione quantistica creato dall'IBM: **Qiskit**. La libreria fornisce svariati strumenti per la creazione e l'esecuzione di programmi quantistici, sia su **simulatori locali** che su **hardware** di dispositivi quantistici offerti dal servizio cloud per la computazione quantistica dell'IBM: **IBM Quantum Experience**. Il vantaggio nell'utilizzo di Qiskit per gli esempi sta nella sua natura autoesplicativa e la facilità di apprendimento per chiunque sia interessato, infatti l'IBM fornisce gratuitamente tutto il materiale necessario: esempi di codice, documentazione e un libro digitale consultabile all'indirizzo <https://qiskit.org/textbook/preface.html>.

1.1 I Qubit

Il **bit** è il concetto fondamentale alla base della computazione classica e analogalmente, nell'ambito dei computer quantistici esiste il **qubit**. Un bit, in ogni momento della computazione, può essere in uno di due stati $\{0, 1\}$ che, rappresentati tramite la notazione **bra-ket** di **Dirac** ' $| \rangle$ ' sono:

$$|0\rangle = \begin{vmatrix} 1 \\ 0 \end{vmatrix} \quad |1\rangle = \begin{vmatrix} 0 \\ 1 \end{vmatrix}$$

Il vettore indica, per il primo bit, che osservandone il valore questo sia sempre 0, viceversa 1 per il secondo. Differentemente da un bit classico, un **qubit** può trovarsi in stati che non siano $|0\rangle$ e $|1\rangle$; esso risiede in uno **spazio di Hilbert bi-dimensionale** nel quale, le basi ortonormali più frequentemente usate sono proprio i vettori $|0\rangle$ e $|1\rangle$. Lo **stato di un qubit** $|\psi\rangle$ è quindi esprimibile come combinazione lineare delle sue basi :

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle = \begin{vmatrix} \alpha \\ \beta \end{vmatrix}$$

dove $|\psi\rangle$ è detta **sovraposizione**, $\begin{vmatrix} \alpha \\ \beta \end{vmatrix}$ è detto **statevector** e $\{\alpha, \beta\}$ numeri complessi. La **probabilità** di misurare 0 e 1 sono rispettivamente $|\alpha|^2$ e $|\beta|^2$, per la regola di **Normalizzazione** dello statevector descritta in seguito.

Mentre è possibile misurare un qubit e verificare se il suo output è 0 o 1, resta impossibile esaminarlo per scoprire il suo stato quantistico, ovvero i valori di α e β poichè, per le leggi della meccanica quantistica, siamo vincolati a poter acquisire solo minime informazioni riguardo lo stato quantistico.

Un qubit può quindi esistere come **combinazione lineare degli stati** $|0\rangle$ e $|1\rangle$, anche se queste non sono le due uniche basi possibili, e l'operazione di misura consiste nel collasso dello stato del qubit in uno dei due stati $|0\rangle$, $|1\rangle$. **Un'altra base** ricorrente in letteratura è quella formata dai qubit $|+\rangle$ e $|-\rangle$, in cui da una misura si ottiene come risultato 0 o 1 il 50% delle volte:

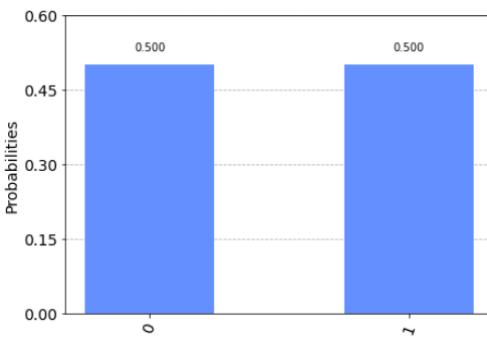
$$|+\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle \quad |-\rangle = \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle$$

Nel seguente blocco di codice vengono stampate le probabilità di ottenere 0 o 1 da un qubit inizializzato nello stato $|+\rangle$:

```

1 backend = Aer.get_backend('statevector_simulator') # simulatore
2 initial_state = [1/sqrt(2), 1/sqrt(2)] # definiamo lo stato iniziale
3 qc = QuantumCircuit(1) # creiamo il circuito quantistico con un qubit in
4 qc.initialize(initial_state, 0) # inizializziamo il qubit con lo stato
5 # definito
6 state = execute(qc,backend).result().get_statevector() # eseguiamo il
7 circuito con il simulatore
8 results = execute(qc,backend).result().get_counts() # salviamo il
9 conteggio dei risultati
10 plot_histogram(results) # stampiamo l'istogramma delle probabilità

```



1.1.1 Regola di Born

Esiste una semplice regola che ci permette di ricavare informazioni da uno stato quantistico senza alcuna misura, detta **regola di Born**, per cui: la probabilità di trovare in uno stato $|x\rangle$ lo stato $|\psi\rangle$ è data da:

$$p(|x\rangle) = |\langle x|\psi\rangle|^2$$

Per la notazione **bra-ket** il valore $\langle x|$ indica il **coniugato complesso** dello stato $|x\rangle$, mentre $\langle x|\psi\rangle$ il **prodotto interno** tra i due stati.

1.1.2 Normalizzazione

Un importante conseguenza della regola di Born è come esponga il legame tra le **ampiezze** $\{\alpha, \beta\}$ e la **probabilità**; se volessimo la sommatoria delle probabilità uguale a uno, come dovrebbe essere, allora lo **statevector** deve essere **normalizzato**, ovvero, il suo modulo deve essere uno:

$$\langle \psi|\psi\rangle = 1$$

segue che: se $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ allora:

$$\sqrt{|\alpha|^2 + |\beta|^2} = 1$$

questo implica che uno stato quantistico deve avere **Norma** $|\alpha|^2 + |\beta|^2 = 1$, perciò, ogni operazione sullo stato deve **preservare questa condizione**, e gli unici operatori in grado di preservare la norma sono le **Matrici Unitarie**.

1.1.3 Sfera di Bloch

La sfera di Bloch è uno strumento per visualizzare la differenza nella **fase relativa** dei vari stati. Dall'equazione di Eulero $z = r^{i\phi}$ sappiamo che la fase di un numero complesso riguarda la rotazione nel piano di Argand, ma non ha alcuna influenza sulle ampiezze dei numeri complessi, poiché dato $z \in \mathbb{C}$ e $r \in \mathbb{R}$, $|z|^2 = |re^{i\phi}|^2 = (re^{i\phi})(r^*e^{i\phi}) = r^2$, la fase ϕ viene ignorata. Le quantità osservabili di uno stato $|\psi\rangle$ sono le probabilità $|\alpha|^2$, $|\beta|^2$ proporzionali alle ampiezze complesse α , β , quindi:

$$|\psi\rangle = r_1 e^{i\phi_1} |0\rangle + r_2 e^{i\phi_2} |1\rangle = e^{i\phi_1} (r_1 |0\rangle + r_2 e^{i(\phi_2 - \phi_1)} |1\rangle)$$

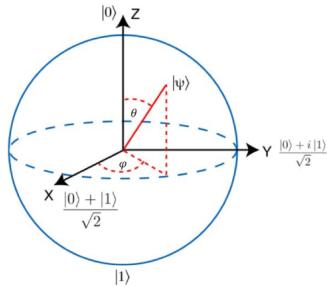
Misurando l'ampiezza $|\psi|^2$ il termine $e^{i\phi_1}$ viene ignorato, questo termine è detto fase globale e non influisce sul calcolo dell'ampiezza, mentre per $\phi = \phi_2 - \phi_1$, $e^{i\phi}$, detta fase relativa permette di riscrivere lo stato $|\psi\rangle$, dati α , $\beta \in \mathbb{R}$, come

$$|\psi\rangle = \alpha |0\rangle + e^{i\phi} \beta |1\rangle$$

Inoltre, basandoci sul fatto che $|\alpha|^2 + |\beta|^2 = 1$, possiamo definire $\alpha = \cos \frac{\theta}{2}$ e $\beta = \sin \frac{\theta}{2}$ da cui segue:

$$|\psi\rangle = \alpha = \cos \frac{\theta}{2} |0\rangle + e^{i\phi} \beta = \sin \frac{\theta}{2} |1\rangle$$

Ora un qubit è rappresentato tramite le variabili $\theta, \phi \in \mathbb{R}$ le quali, interpretate come coordinate sferiche, permettono di visualizzarlo sulla sfera di Bloch:



Agli estremi dell'asse x ci sono gli stati $|+\rangle$ e $|-\rangle$, su quella z gli stati $|0\rangle$ e $|1\rangle$ e sull'asse delle y gli stati $|R\rangle$ e $|L\rangle$, ovvero:

$$|R\rangle = \frac{1}{\sqrt{2}} |0\rangle + i \frac{1}{\sqrt{2}} |1\rangle \quad |L\rangle = \frac{1}{\sqrt{2}} |0\rangle - i \frac{1}{\sqrt{2}} |1\rangle$$

Vedremo poi come l'applicazione di **Gate Quantistici** corrisponda a una rotazione del vettore, corrispondente allo stato del qubit, sulla sfera di Bloch.

1.2 Computazione Quantistica

1.2.1 Quantum Gates

Analogalmente ai gate logici dei computer classici, nella computazione quantistica, l'informazione è manipolata tramite **gate quantistici**, i quali conservano il numero di qubits, al contrario di gate classici come l'AND o l'OR che non lo fanno. La conservazione del numero di qubit, deriva dall'unico vincolo presente sui gate quantistici: la **condizione di normalizzazione** per uno stato $|\psi\rangle = \alpha |0\rangle + \beta |1\rangle$ deve restare valida anche nello stato in cui il qubit si troverà dopo l'applicazione del gate $|\psi'\rangle = \alpha' |0\rangle + \beta' |1\rangle$, di conseguenza la **matrice associata al gate** U deve essere unitaria: $UU^\dagger = I$. Questa **condizione di unitarietà** rende quindi possibile per ogni matrice unitaria di essere un valido gate quantistico.

1.2.2 Gate per singoli qubit (Hermitian)

I **gate di Pauli** sono i più utilizzati, ed equivalgono a una rotazione di π intorno alle assi x, y, z della sfera di Bloch:

$$X = \begin{vmatrix} 0 & 1 \\ 1 & 0 \end{vmatrix}, \quad Y = \begin{vmatrix} 0 & -i \\ i & 0 \end{vmatrix}, \quad Z = \begin{vmatrix} 1 & 0 \\ 0 & 1 \end{vmatrix}$$

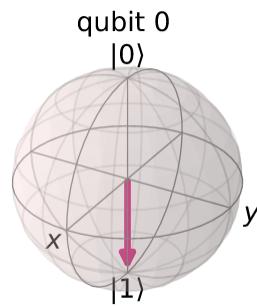
Ad esempio l'effetto dell' X gate corrisponde al NOT, che è in grado di ruotare il vettore sulle sfera di Bloch dallo stato $|0\rangle$ allo stato $|1\rangle$ e viceversa.

Ad esempio applicato X allo stato $|0\rangle$

```

1 qc = QuantumCircuit(1)
2 qc.x(0)
3 qc.draw()
4 backend = Aer.get_backend('statevector_simulator')
5 out = execute(qc,backend).result().get_statevector()
6 plot_bloch_multivector(out)

```



Gli stati su cui l'applicazione del gate non ha effetto sono detti **Autostati** del gate.

ad esempio applicare lo Z gate alle basi $|0\rangle, |1\rangle$ non ha effetto, poiche questi si trovano già sull'asse Z, allo sessto modo X per $|+\rangle, |-\rangle$ e Y per $|R\rangle, |L\rangle$

Gli operatori di Pauli, se generalizzati, ci portano a un operatore unitario $R(\theta)_{\{X,Y,Z\}}$ che effettua, sull'asse associata, una rotazione di angolo θ , anzichè essere vincolata a π :

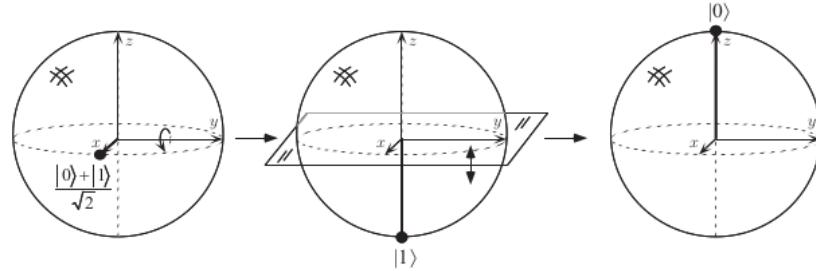
$$R_X(\theta) = e^{-i\theta X/2} = \cos \frac{\theta}{2} I - i \sin \frac{\theta}{2} X = \begin{bmatrix} \cos \frac{\theta}{2} & -i \sin \frac{\theta}{2} \\ -i \sin \frac{\theta}{2} & \cos \frac{\theta}{2} \end{bmatrix}$$

$$R_Y(\theta) = e^{-i\theta Y/2} = \cos \frac{\theta}{2} I - i \sin \frac{\theta}{2} Y = \begin{bmatrix} \cos \frac{\theta}{2} & -\sin \frac{\theta}{2} \\ \sin \frac{\theta}{2} & \cos \frac{\theta}{2} \end{bmatrix}$$

$$R_Z(\theta) = e^{-i\theta Z/2} = \cos \frac{\theta}{2} I - i \sin \frac{\theta}{2} Z = \begin{bmatrix} \cos \frac{\theta}{2} - i \sin \frac{\theta}{2} & 0 \\ 0 & \cos \frac{\theta}{2} + i \sin \frac{\theta}{2} \end{bmatrix}$$

Altri gate tipici sono: l'**Hadamard H**, l'**Identità I**, i gate S e S^t e infine il gate T . L' Hadamard gate effettua un cambio di basi tra x -basis e z -basis, permette cioè di passare da uno dei due poli della sfera di Bloch a una sovrapposizione, esso non è altro che una rotazione sull'asse y di 90° seguita da una sull'asse x di 180° :

$$H = \frac{1}{\sqrt{2}} \begin{vmatrix} 1 & 1 \\ 1 & -1 \end{vmatrix},$$



I gate S e S^t , anche detti $R(\frac{\pi}{2})_Z$ e $R(-\frac{\pi}{2})_Z$ o anche \sqrt{Z} e \sqrt{Z}^t poiché $SS|q\rangle = Z|q\rangle$:

$$S = \begin{vmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{2}} \end{vmatrix}, \quad S^t = \begin{vmatrix} 1 & 0 \\ 0 & e^{-i\frac{\pi}{2}} \end{vmatrix}$$

Infine il T -gate, conosciuto anche come $R(\frac{\pi}{4})_Z$ o $\sqrt[4]{Z}$, e il suo trasposto T^t

$$T = \begin{vmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{vmatrix} \quad T^t = \begin{vmatrix} 1 & 0 \\ 0 & e^{-i\frac{\pi}{4}} \end{vmatrix}$$

Tutti i gate sopra nominati posso essere **generalizzati** nell' U_3 -gate:

$$U_3(\theta, \phi, \lambda) = \begin{vmatrix} \cos \frac{\theta}{2} & -e^{\lambda i} \sin \frac{\theta}{2} \\ -e^{\phi i} \sin \frac{\theta}{2} & -e^{\lambda i + \phi i} \cos \frac{\theta}{2} \end{vmatrix}$$

1.2.3 Gate per singoli qubit su statevector multipli

Per descrivere lo **stato di due qubit** abbiamo bisogno di uno state vector 4-dimensionale:

$$|a\rangle = a_{00}|00\rangle + a_{01}|01\rangle + a_{10}|10\rangle + a_{11}|11\rangle = \begin{vmatrix} \mathbf{a}_{00} \\ \mathbf{a}_{01} \\ \mathbf{a}_{10} \\ \mathbf{a}_{11} \end{vmatrix}$$

ovviamente la regola di Born e quella di normalizzazione restano valide:

$$p(|00\rangle) = |\langle 00|a\rangle|^2 = |a_{00}|^2$$

$$|a_{00}|^2 + |a_{01}|^2 + |a_{10}|^2 + |a_{11}|^2 = 1$$

inoltre, dati due o più qubit, è possibile rappresentare il loro stato composto tramite il loro **prodotto tensoriale**

$$|a\rangle = \begin{vmatrix} \mathbf{a}_0 \\ \mathbf{a}_1 \end{vmatrix} \quad |b\rangle = \begin{vmatrix} \mathbf{b}_0 \\ \mathbf{b}_1 \end{vmatrix}$$

$$|ba\rangle = |b\rangle \otimes |a\rangle = \begin{vmatrix} \mathbf{b}_0 \mathbf{a}_0 \\ \mathbf{b}_0 \mathbf{a}_1 \\ \mathbf{b}_1 \mathbf{a}_0 \\ \mathbf{b}_1 \mathbf{a}_1 \end{vmatrix}$$

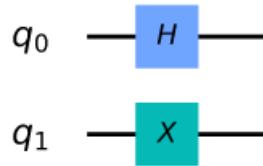
Possiamo quindi rappresentare l'azione di **più operatori su più qubit** tramite una singola moltiplicazione tra matrici: moltiplicando il prodotto tensoriale degli operatori con il prodotto tensoriale degli statevector.

Ad esempio per il seguente circuito

```

1 qc = QuantumCircuit(2)
2 qc.h(0)
3 qc.x(1)
4 qc.draw()

```



$$X|q_1\rangle \otimes H|q_0\rangle = (X \otimes H)|q_0 q_1\rangle$$

1.2.4 Gate per qubit multipli

Solitamente i gate applicati su più di un qubit poggiano su due insiemi: uno detto di **controllo** e un'altro detto **obiettivo**; il caso più semplice è il **Controlled-Not**. Il CNOT-gate su due qubit ha l'effetto di effettuare un X -gate sul qubit obiettivo se e solo se il controllo è 1:

```

1 qc = QuantumCircuit(2)
2 qc.cx(0,1)
3 qc.draw()

```



L'effetto può essere descritto come $|q_0, q_1\rangle \implies |q_0, q_0 \oplus q_1\rangle$ ed è rappresentato dalla matrice:

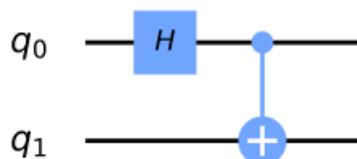
$$U_{CN} = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{vmatrix}$$

È chiaro come il CNOT agisca sugli stati classici $\{|0\rangle, |1\rangle\}$, ma su qubit in sovrapposizione l'effetto è meno intuitivo, infatti l'applicazione ci permette di creare uno stato detto **Entangled**:

```

1 qc = QuantumCircuit(2)
2 qc.h(0)
3 qc.cx(0,1)
4 qc.draw()

```



Questo circuito genera lo statevector $\begin{vmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ 0 \\ \frac{1}{\sqrt{2}} \end{vmatrix}$, il quale indica che esiste un 50% di probabilità di misurare $|00\rangle$ e un 50% di misurare $|11\rangle$

$$CNOT|+0\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$$

Lo stato così generato è uno dei quattro **stati di Bell**, la sua particolarità risiede nel fatto che è uno **stato combinato** non scindibile in due stati distinti. Nonostante siano in sovrapposizione, misurare uno dei due qubit ci dirà lo stato dell'altro facendone collassare la sovrapposizione, questo per via del cosiddetto **entanglement quantistico** creatosi tra i due qubit; anche allontanandoli di anni luce questo legame è inscindibile, e la misura di uno provoca il collasso dell'altro istantaneamente. Purtroppo non è possibile sfruttare l'entanglement per comunicare più velocemente della luce, poiché il risultato di una misura è comunque casuale, come viene dimostrato in un teorema detto "**Teorema della non-comunicazione**".

2 Portata della computazione quantistica

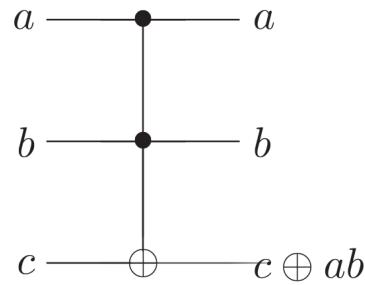
fonti bibliografiche: [1].

2.1 Computazione classica su computer quantistici

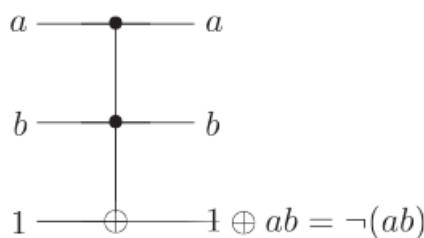
È possibile simulare un circuito logico classico su un circuito quantistico?

La ragione per la quale i circuiti quantistici non possono simulare quelli logici, sta nel fatto che i **gate unitari sono sempre reversibili** mentre molti gate logici, come ad esempio il NAND, sono irreversibili, di conseguenza non sono rappresentabili da matrici unitarie. Però ogni circuito classico può essere rimpiazzato da uno equivalente, contentente solo elementi reversibili, mediante un gate reversibile detto **Toffoli-gate**. Il Toffoli-gate ha 3 bit in input e 3 in output, due servono da controllo, mentre il terzo è l'obbiettivo, il cui valore viene negato se e solo se entrambi i bit di controllo sono a 1, inoltre, applicarlo due volte consecutive ha l'effetto $(a, b, c) \rightarrow (a, b, c \oplus ab) \rightarrow (a, b, c)$, da cui segue che è un gate reversibile.

Inputs			Outputs		
a	b	c	a'	b'	c'
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	1	0	1
1	1	0	1	1	1
1	1	1	1	1	0



Ad esempio può essere usato per simulare l'operazione di NAND:



Anche se descritto come un gate classico, può essere facilmente implementato come uno quantistico, ovvero tramite un CNOT con due qubit di controllo, il quale viene rappresentato da una matrice unitaria 8×8 , utilizzata poi per **simulare tutti i gate irreversibili classici**; in questo modo un computer quantistico è in grado di simulare ogni computazione che un computer classico è in grado di eseguire.

Ovviamente la capacità di simulare un computer classico **non è efficiente**, poiché si deve tener conto degli effetti del **rumore** quantistico; il vantaggio della computazione quantistica non risiede infatti nella simulazione di sistemi classici, ma sta nel fatto che i computer quantistici sono in grado di computare funzioni molto più complesse e questo ci dà l'opportunità di sviluppare **algoritmi quantistici** in grado di risolvere alcune classi di problemi molto più velocemente di un computer classico.

2.2 La classe P, NP, PSPACE e BQP

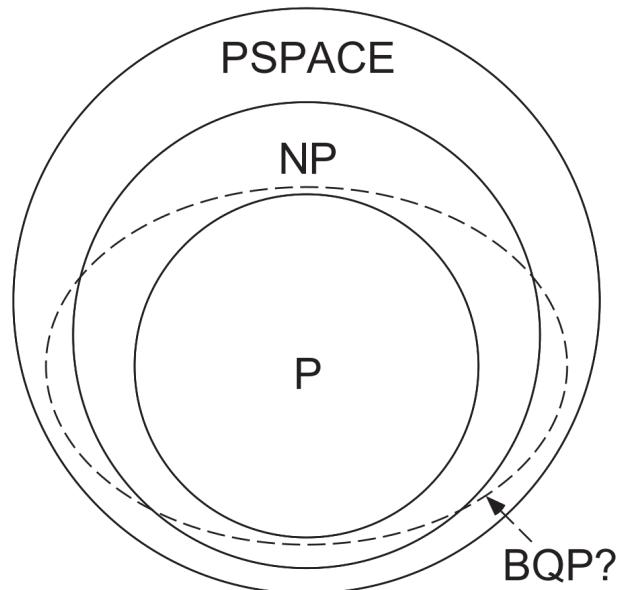
La **teoria della complessità** classifica la difficoltà computazionale di vari problemi, questi sono divisi in classi di complessità in base a caratteristiche comuni riguardanti le risorse computazionali utilizzate per risolverli. Le due classi più importanti della teoria sono la classe **P** e **NP**, in breve: **P (polynomial)** sono i problemi la cui complessità è esprimibile come polinomio dipendente dalla grandezza dell'input, e sono quindi velocemente risolvibili su un computer classico, mentre la complessità dei problemi in **NP (non polynomial)**, non è esprimibile come polinomio, di conseguenza non sono velocemente risolvibili, ma le soluzioni di tali problemi possono essere facilmente verificate su un computer classico.

È chiaro che $P \subseteq NP$ poiché l'abilità di risolvere un problema implica la possibilità di verificarne le soluzioni, ma non è ancora chiaro se esistono problemi $p \in NP$ tali che $p \notin P$, infatti il problema più importante nella teoria della complessità è stabilire se queste due classi siano equivalenti, ovvero $P \stackrel{?}{=} NP$.

Esiste una sottoclasse dei problemi NP, i problemi **NP-completi**, importanti per il fatto che esiste una riduzione, eseguibile in tempo polinomiale, da ogni problema NP ad ogni problema NP-completo, perciò se un singolo problema NP-completo p fosse risolvibile in tempo polinomiale $p \in P$ allora lo sarebbero tutti i problemi in NP, con la conseguenza che $P = NP$.

Un'altra importante classe di problemi è **PS (polynomial space bounded)**, ovvero quei problemi risolvibili con una limitazione nello spazio di memoria utilizzato (ma non necessariamente nel tempo che impiega), si crede, ma non è dimostrato, che questa classe sia leggermente più grande di NP (e quindi anche di P).

Infine la classe **BQP** è l'insieme di problemi risolvibili efficientemente da un computer quantistico. Ancora non è certo come si collochi BQP rispetto P, NP e PS; tutto ciò che finora sappiamo, è che un computer quantistico può risolvere efficientemente ogni problema in P, ma non problemi fuori da PS; segue che BQP giace tra P e PS. Un implicazione importante è che, se fosse dimostrato che i computer quantistici sono più potenti dei computer classici, allora seguirebbe che $P \neq PS$.



La classe **BQP** è in realtà computazionalmente più simile a **BPP (Bounded probability polynomial)** che a **P**, dove BPP è la classe di problemi risolvibili con **algoritmi randomizzati** con limite sulla **probabilità di errore**, la quale deve restare inferiore a $\frac{1}{3}$, infatti, parlando di computer quantistici la probabilità di errore è da tenere in considerazione per via del rumore nell'hardware quantistico.

È importante sottolineare che nonostante sia possibile creare una sovrapposizione uniforme di stati, le regole di misura implicano che utilizzando solamente questa forma facilmente ottenibile di sovrapposizione non è comunque possibile la risoluzione di problemi NP-completi, come ad esempio **SAT (Satisfiability Problem)**, cioè il problema di decidere se una **formula booleana sia soddisfacibile** o meno.

Se un'istanza di SAT di lunghezza n , è risolta da p assegnamenti su 2^n , allora la probabilità di trovare la soluzione dopo una sola misura è $\frac{p}{2^n}$: non abbiamo fatto altro che prendere casualmente un assegnamento nella speranza che soddisfi la formula. Per questo motivo la risoluzione di SAT in tempo polinomiale su un computer quantistico non è creduto possibile e la maggior parte dei ricercatori credono fortemente che $NP \not\subseteq BQP$, assunzione che, se dimostrata risolverebbe il problema già citato $P \neq NP$.

3 Algoritmi quantistici

fonti bibliografiche: [1], [2].

3.1 Classi di algoritmi quantistici

Esistono principalmente **3 classi** di algoritmi quantistici che forniscono un vantaggio computazionale rispetto alle note controparti classiche. Per prima c'è una classe di algoritmi basati su una versione quantistica della trasformata di Fourier, che comprende, ad esempio: l'algoritmo di Deutsch-Josza e quello di Shor per la fattorizzazione. La seconda classe riguarda algoritmi per la ricerca quantistica, come l'algoritmo di Grover. Infine, la terza comprende gli algoritmi usati per la simulazione quantistica .

3.2 Algoritmi quantistici basati sulla trasformata di Fourier

3.2.1 Trasformata quantistica di Fourier

La trasformata quantistica di Fourier, **QFT**, è un cambio di basi, da basi computazionali a basi di Fourier.

Per 1 qubit con basi $\{|0\rangle, |1\rangle\}$, le basi di Fourier sono $\{|-\rangle, |+\rangle\}$, segue che in questo caso la QFT equivale all'applicazione di un Hadamard-gate

Dato $N=2^n$, poichè per n qubit ci sono 2^n stati base:

$$|\tilde{x}\rangle = QFT|x\rangle = \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} e^{\frac{2\pi i xy}{N}} |y\rangle$$

dove $|\tilde{x}\rangle$ è la base di Fourier e $|x\rangle$ la base computazionale.

È banale verificare che $QFT|0\rangle = |+\rangle$ e che $QFT|1\rangle = |-\rangle$

L'equazione, così com'è, non dà indicazioni sulla struttura del circuito quantistico che la implementi, va quindi manipolata. Introduciamo la seguente notazione per la rappresentazione binaria:

$$y = [y_1, y_2, \dots, y_n] = 2^{n-1} + 2^{n-2} + \dots + 2^{n-0} = \sum_{k=1}^n y_k 2^{n-k}$$

Dove, ad esempio $y = 5 \rightarrow [101]$

Ricordiamo inoltre che:

$$\sum_{y=0}^{N-1} = \sum_{y_1=0}^1 \sum_{y_2=0}^1 \dots \sum_{y_n=0}^1$$

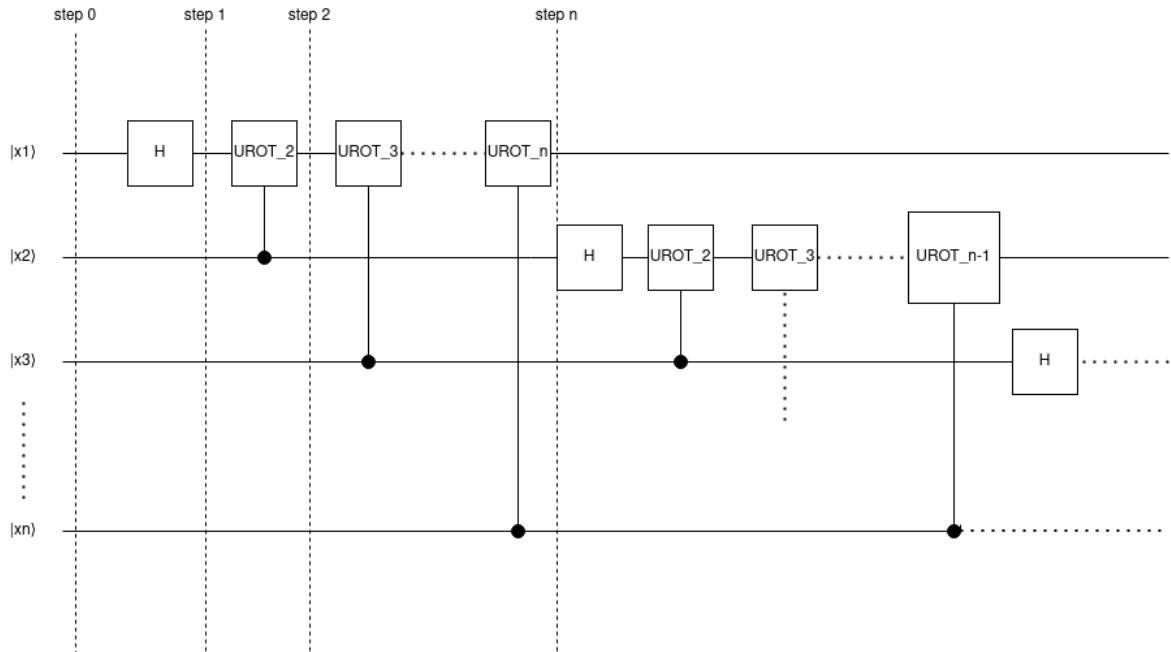
Questo ci permette di riscrivere l'equazione della QFT come:

$$\begin{aligned} |\tilde{x}\rangle &= QFT|x\rangle = \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} e^{2\pi i x \sum_{k=1}^n y_k} |y\rangle = \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} \prod_{k=1}^n e^{\frac{2\pi i x y_k}{2^k}} |y\rangle = \\ &= \frac{1}{\sqrt{N}} (|0\rangle + e^{\frac{2\pi i x}{2^1}} |1\rangle) \otimes (|0\rangle + e^{\frac{2\pi i x}{2^2}} |1\rangle) \otimes (|0\rangle + e^{\frac{2\pi i x}{2^3}} |1\rangle) \otimes \dots \otimes (|0\rangle + e^{\frac{2\pi i x}{2^n}} |1\rangle) \end{aligned}$$

Ora il pattern risulta più chiaro, ovvero preso x_k dallo stato $|x\rangle = |x_1 x_2 \dots x_n\rangle = |x_1\rangle \otimes |x_2\rangle \otimes \dots \otimes |x_n\rangle$ questo viene trasformato dalla QFT in $(|0\rangle + e^{\frac{2\pi i x_k}{2^k}} |1\rangle)$, e il prodotto tensoriale degli elementi è scalato di $\frac{1}{\sqrt{N}}$. Per effettuare questa trasformazione su un singolo qubit basta un Hadamard gate, poiché, se $x_k = 0$ allora $e^{\frac{2\pi i x_k}{2^k}} = 1$ altrimenti per $x_k = 1$ segue $e^{\frac{2\pi i x_k}{2^k}} = -1$. Per la costruzione del circuito quantistico che realizza la QFT è necessario anche un operatore detto **Unitary Rotation** $UROT_K$, il quale applicato allo stato $|0\rangle$ lo lascia invariato, mentre applicato a $|1\rangle$ restituisce $e^{\frac{2\pi i}{2^k}} |1\rangle$, ovvero applica allo stato la fase $e^{\frac{2\pi i}{2^k}}$. Per il circuito utilizzeremo la sua versione controllata: C_{UROT_k} (Da notare che $UROT_1 = H$).

$$UROT_k = \begin{vmatrix} \mathbf{1} & 0 \\ \mathbf{0} & e^{\frac{2\pi i}{2^k}} \end{vmatrix},$$

Il circuito che implementa la QFT è il seguente:



Analizzandone i vari step:

step 0: $|x_1 x_2 \dots x_m\rangle$

step 1: $(|0\rangle + e^{\frac{2\pi i x_1}{2^1}} |1\rangle) \otimes |x_2 \dots x_n\rangle$

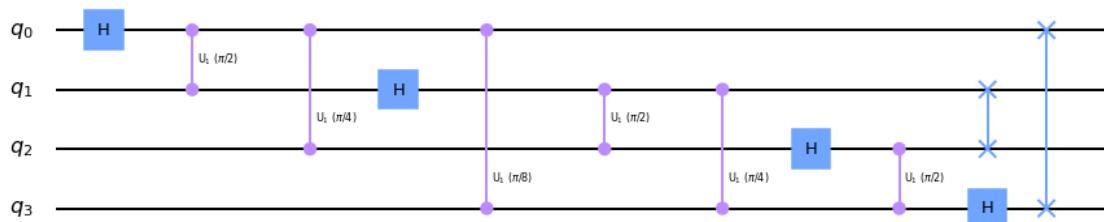
step 2: $(|0\rangle + e^{\frac{2\pi i x_1}{2^2}} + e^{\frac{2\pi i x_2}{2^2}} |1\rangle) \otimes |x_2 \dots x_n\rangle$

...

step n: $(|0\rangle + e^{\frac{2\pi i x_n}{2^n}} + \dots + e^{\frac{2\pi i x_1}{2^n}} + e^{\frac{2\pi i x_2}{2^n}} + \dots + e^{\frac{2\pi i x_n}{2^n}} |1\rangle) \otimes |x_2 \dots x_n\rangle$

Il circuito realizza la QFT, ma con i qubit in ordine inverso; in Qiskit è presente il circuito che realizza lo swap dei qubit.

Ad esempio per 4 qubit:



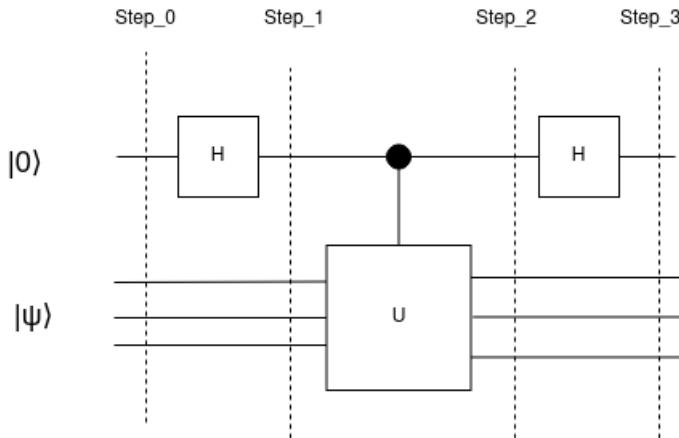
3.2.2 Stima della fase quantistica (QPE)

Un'applicazione della QFT è la **QPE (Quantum Phase Estimation)**, la cui applicazione è rilevante per la **simulazione quantistica**, dove l'evoluzione dell'Hamiltoniana che descrive il sistema (cioè l'evoluzione nel tempo di sistemi reali) è descritta da una trasformazione unitaria.

Ricordando che una **matrice unitaria** U ha **autovalori** della forma $e^{i\theta}$, se ha **autovettori** che formano una **base ortonormale** allora:

$$U|\psi\rangle = e^{i\theta_\psi} |\psi\rangle$$

La QPE permette di estrarre la fase θ_ψ avendo l'abilità di preparare lo stato $|\psi\rangle$ e quella di applicare la trasformazione U . Il circuito è il seguente:



Analizzandone i vari step:

$$\text{step 0: } |0\rangle|\psi\rangle$$

$$\text{step 1: } \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)|\psi\rangle$$

$$\text{step 2: } \frac{1}{\sqrt{2}}(|0\rangle|\psi\rangle + |1\rangle e^{i\theta_\psi} |\psi\rangle)$$

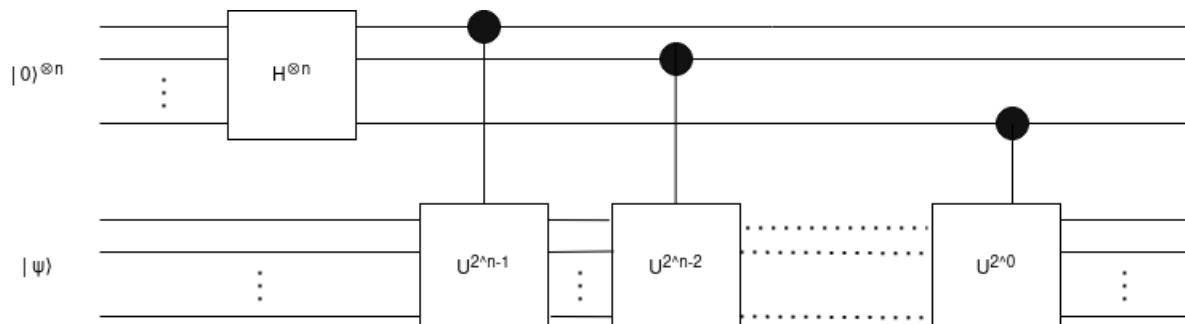
$$\text{step 3: } \frac{1}{\sqrt{2}}\left(\left(\frac{|0\rangle+|1\rangle}{\sqrt{2}}\right)|\psi\rangle + e^{i\theta_\psi}\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right)|\psi\rangle\right) = \frac{1}{2}[|0\rangle(1 + e^{i\theta_\psi}) + |1\rangle(1 - e^{i\theta_\psi})]|\psi\rangle$$

Segue che:

$$p(0) = \left|\frac{1}{2}(1 + e^{i\theta_\psi})\right|^2$$

$$p(1) = \left|\frac{1}{2}(1 - e^{i\theta_\psi})\right|^2$$

Eseguendo il circuito ripetutamente, le occorrenze di $|0\rangle$ e $|1\rangle$ nell'output ci permetteranno una stima di θ_ψ , la precisione di tale stima può essere migliorata utilizzando più qubit, come nel seguente circuito:



Ricordando che: $U^{2^x}|\psi\rangle = U^{2^{x-1}}e^{i\theta_\psi}|\psi\rangle = U^{2^{x-2}}e^{i\theta_\psi}e^{i\theta_\psi}|\psi\rangle$ (e così via fino a consumare U^{2^0}), esaminiamo i vari step.

step 0: $|0\rangle^{\otimes n}|\psi\rangle$

step 1: $(\frac{1}{\sqrt{2}})^n(|0\rangle + |1\rangle)^{\otimes n}|\psi\rangle$

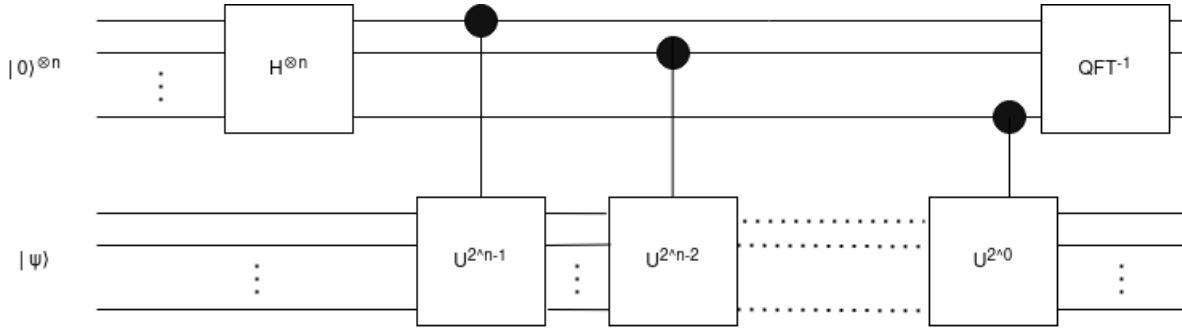
...

step f: $(\frac{1}{\sqrt{2}})^n(|0\rangle + e^{i\theta_\psi 2^{n-1}}|1\rangle) \otimes (|0\rangle + e^{i\theta_\psi 2^{n-2}}|1\rangle) \otimes \dots \otimes (|0\rangle + e^{i\theta_\psi 2^0}|1\rangle)$

Si vede ora che l'output del circuito **equivale alla QFT**

$|\tilde{x}\rangle = \frac{1}{\sqrt{N}}(|0\rangle + e^{\frac{2\pi ix}{2^1}}|1\rangle) \otimes (|0\rangle + e^{\frac{2\pi ix}{2^2}}|1\rangle) \otimes (|0\rangle + e^{\frac{2\pi ix}{2^3}}|1\rangle) \otimes \dots \otimes (|0\rangle + e^{\frac{2\pi ix}{2^n}}|1\rangle)$, ma con una

fase diversa, poichè nella QFT: $\theta_\psi = \frac{\theta_\psi 2\pi}{2^n}$. Grazie a questa differenza, applicando QFT^{-1} al circuito della QPE, in output ci troveremo lo stato $|2^n\theta_\psi\rangle$ da cui è possibile stimare la fase (Da ricordare che dati dei gate unitari A, B, C segue che $(ABC)^{-1} = C^{-1}B^{-1}A^{-1}$).



3.2.3 Algoritmo di Shor

Il problema affrontato dall'algoritmo di Shor è la **fattorizzazione di interi** del tipo $N = p * q$, dove p, q sono primi particolarmente grandi. La difficoltà per un computer classico di affrontare il problema è alla base degli algoritmi di cifratura moderni per la crittografia su larga scala, come ad esempio l' algoritmo **RSA** (Rivest, Shamir, Adleman) poichè la **complessità classica** di tale problema è $O(e^{c*n^{\frac{1}{3}}(\log(n))^{2/3}})$ dove n è il numero di bit usato per i fattori. L'algoritmo di Shor fornisce uno **speed up-esponenziale**, portando la complessità a $O(n^3)$, sfruttando la QFT e la QPE.

Protocollo

1. Prendiamo un numero a co-primo con N ;
2. troviamo il periodo r della funzione $a^r mod(N)$, ovvero il più piccolo r tale che $a^r = 1 (mod N)$;
3. se r è pari, scegliamo un x tale che $x \equiv a^{r/2} mod(N)$;
 1. se $x + 1 \not\equiv 0 mod(N)$ allora almeno uno tra $\{p, q\}$ è contenuto in $\{gcd(x + 1, N), gcd(x - 1, N)\}$;
 2. altrimenti troviamo un altro a ;
4. altrimenti, se r è dispari, troviamo un altro a ;

Ad esempio, preso $N=15$:

1. troviamo un co-primo di 15, come $a = 13$

2. troviamo il periodo di $13^r mod(15)$, cioè $r = 4$, poichè:

$$x = 0, 1, 2, 3, 4, \dots$$

$$13^x mod(15) = 1, 13, 4, 7, 1, \dots$$

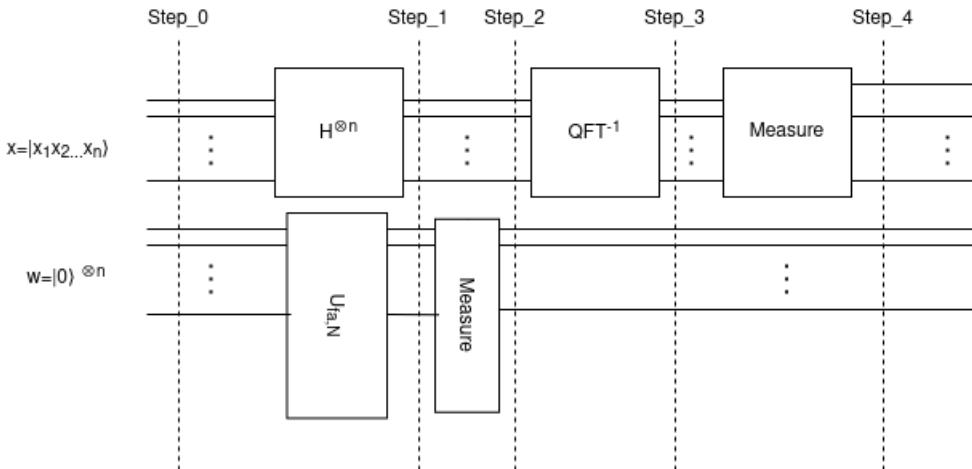
3. $x = a^{r/2} mod(N) = 13^{4/2} mod(15) = 4 mod(15)$

4. verifichiamo se $x + 1 \not\equiv 0 \text{ mod}(N)$;

$x + 1 = 4 + 1 = 5 \equiv 5 \text{ mod}(15) \not\equiv 0 \text{ mod}(15)$, di conseguenza almeno uno tra $\{p, q\}$ è contenuto in: $\{\gcd(x+1, N), \gcd(x-1, N)\} = \{\gcd(5, 15), \gcd(3, 15)\} = \{5, 3\}$. Essendo $\{5, 3\}$ gli unici due primi nell'intervallo, segue che $p = 3, q = 5$.

Circuito quantistico per la fattorizzazione di $N=pq$

Il seguente circuito implementa la fattorizzazione precedentemente descritta: n rappresenta il numero di bit usati per rappresentare N , mentre il gate unitario $U_{f_{a,n}}$ implementa la funzione $f_{a,n}(x) = a^x \text{ mod}(N)$, che effettua l'operazione $|x\rangle|w\rangle \xrightarrow{} |x\rangle|w \oplus f_{a,n}(x)\rangle$, inoltre, l'output del circuito viene dato in input a del **post-processing classico**, corrispondente al punto 3 del protocollo.



Ad esempio, preso $N=15$, come nel precedente esempio, analizziamo i vari step del circuito:

$$step_0 : |x\rangle|w\rangle = |0\rangle^{\otimes 4}|0\rangle^{\otimes 4}$$

$$step_1 : [H^{\otimes 4}|0\rangle^{\otimes 4}]|0\rangle^{\otimes 4} = \frac{1}{4}[|0\rangle_4 + |1\rangle_4 + \dots + |15\rangle_4]|0\rangle_4 = \frac{1}{4}[|0000\rangle + |0001\rangle + \dots + |1111\rangle]|0\rangle_4$$

$$step_2 : \frac{1}{4}[|0\rangle_4|0\rangle_4 \oplus 13^0 \text{ mod}(15)\rangle_4 + |1\rangle_4|0\rangle_4 \oplus 13^1 \text{ mod}(15)\rangle_4 + \dots] =$$

$$\begin{aligned} &= \frac{1}{4}[|0\rangle_4|1\rangle_4 + |1\rangle_4|13\rangle_4 + |2\rangle_4|4\rangle_4 + |3\rangle_4|7\rangle_4 + \\ &\quad + |4\rangle_4|1\rangle_4 + |5\rangle_4|13\rangle_4 + |6\rangle_4|4\rangle_4 + |7\rangle_4|7\rangle_4 + \\ &\quad + |8\rangle_4|1\rangle_4 + |9\rangle_4|13\rangle_4 + |10\rangle_4|4\rangle_4 + |11\rangle_4|7\rangle_4 + \\ &\quad + |2\rangle_4|1\rangle_4 + |13\rangle_4|13\rangle_4 + |14\rangle_4|4\rangle_4 + |15\rangle_4|7\rangle_4] \end{aligned}$$

Nello step 2 il pattern è chiaro, da questo possiamo estrapolare $|x\rangle$ partendo dall'output della misurazione di $|w\rangle$, se ad esempio $|w\rangle = |7\rangle_4$ allora $|x\rangle$ può prendere uno tra i valori $|3\rangle_4, |7\rangle_4, |11\rangle_4, |15\rangle_4$. È importante notare il cambio di normalizzazione, prima avevamo 16 possibili output, perciò la probabilità di ognuno era $\frac{1}{16}$ mentre ora ne abbiamo quattro, quindi è $\frac{1}{4}$:

$$step_3 : |x\rangle|w\rangle = \frac{1}{2}[|3\rangle_4 + |7\rangle_4 + |11\rangle_4 + |15\rangle_4] \otimes |7\rangle_4$$

Ora dobbiamo calcolare la $QFT^t|x\rangle$, perciò dobbiamo ricavarci le varie trasformate trasposte dei suoi componenti.

$$step_4 : \frac{1}{8} \sum_{y=0}^{15} [e^{-\frac{-i3\pi}{8}y} + e^{\frac{-i7\pi}{8}y} e^{\frac{-i11\pi}{8}y} e^{\frac{-i15\pi}{8}y}] = \frac{1}{8}[4|0\rangle_4 + 4i|4\rangle_4 - 4|8\rangle_4 - 4i|12\rangle_4]$$

Misuriamo a questo punto il registro $|x\rangle$

$$step_5 : x = \{0, 4, 8, 12\}$$

Rimane solo da fare del post processing, ovvero il terzo punto del protocollo descritto: a seconda dell'output potremmo ottenere una soluzione totale o parziale, cioè entrambi o uno solo dei fattori, nel secondo caso, da un fattore è banale trovare il secondo.

Ricordando che stiamo cercando il periodo r e che i risultati delle misure raggiungono un picco in $j \frac{N}{r}$ per $j \in Z$:

Per, ad esempio, l'output $|4\rangle_4$, segue che, l'equazione $j \frac{16}{r} = 4$ è verificata per $j = 1$, $r = 4$, e dato che r è pari: $x \equiv a^{\frac{r}{2}} \text{ mod}(N) = 13^{\frac{4}{2} \text{ mod}(15)} = 4$, quindi $\gcd(x+1, N) = \gcd(5, 15) = 5$ e $\gcd(x-1, N) = \gcd(3, 15) = 3$. Abbiamo trovato i fattori $\{3, 5\}$ per $N = 15$

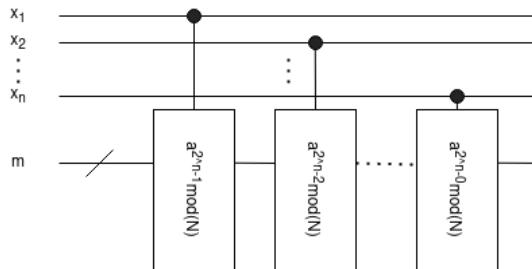
Prendendo, invece, l'output $|8\rangle_4$, segue che, l'equazione, $j \frac{16}{r} = 8$ è verificata per le coppie di valori: $j = 1$, $r = 2$ e $j = 2$, $r = 4$. Preso il primo caso, r è pari e $x \equiv a^{\frac{r}{2}} \text{ mod}(N) = 13^{\frac{2}{2} \text{ mod}(15)} = 2$, quindi: $\gcd(x+1, N) = \gcd(3, 15) = 3$ e $\gcd(x-1, N) = \gcd(1, 15) = 1$. Escludendo $p = 1$ come soluzione, perchè banale, ci resta la soluzione parziale $p = 3$, da cui ricaviamo l'altro fattore $q = \frac{N}{p} = \frac{15}{3} = 5$.

Da notare che se esce $|0\rangle_4$ dobbiamo provare ancora l'esecuzione del circuito.

L'ultima cosa da aggiungere è come viene implementato il circuito per $U_{f_{a,n}}$. Ricordando che $x = [x_1, x_2, \dots, x_n] = 2^{n-1}x_1 + 2^{n-2}x_2 + \dots + 2^0x_n$ allora:

$$f_{a,n}(x) = a^x \text{ mod}(N) = a^{2^{n-1}x_1 + 2^{n-2}x_2 + \dots + 2^0x_n} \text{ mod}(N) = a^{2^{n-1}x_1}a^{2^{n-2}x_2}\dots a^{2^0x_n} \text{ mod}(N)$$

Il circuito per implementare la funzione è il seguente:



Si vede facilmente che il pattern del **circuito è lo stesso usato in QPE**, per questo si può affermare che l'algoritmo di **Shor è una QPE sotto mentite spoglie**, dal fatto che sfrutta lo stesso gate $U^{2^x} = a^x \text{ mod}(N)$.

3.3 Algoritmi quantistici di ricerca

Una classe completamente differente di algoritmi è quella degli **algoritmi di ricerca quantistica**, le cui basi sono state gettate da **Lov Grover** con il suo omonimo algoritmo. Questa classe di algoritmi risolve il seguente **problema**: *dato uno spazio di ricerca di dimensione N , e nessuna conoscenza a priori sulla struttura dell'informazione in esso contenuta, vogliamo trovare un elemento che soddisfa una data proprietà*.

La versione classica di questo problema è risolta con, al più, N operazioni, ma la versione quantistica di Grover ci permette di utilizzarne approssimativamente \sqrt{N} . Di certo uno **speed-up quadratico** non è nulla a confronto di quelli esponenziali offerti dagli algoritmi basata sulla QFT, ma ad ogni modo è più utile poichè le euristiche di ricerca hanno un notevole utilizzo in svariate applicazioni.

3.4 Algoritmi per la simulazione quantistica

La **simulazione di sistemi quantistici** appare come un **naturale candidato** per l'utilizzo di computer quantistici, mentre è ritenuta difficile per i computer classici. La difficoltà nell'affrontare tale problema ha radici nella stessa motivazione per cui è complicato simulare i computer quantistici: la quantità di numeri complessi da utilizzare per descrivere un sistema quantistico cresce esponenzialmente con la grandezza del sistema, anzichè linearmente come in uno classico.

Generalmente immaginare lo stato quantistico di un sistema con n componenti distinti, prende c^n bit di memoria su un computer classico, dove c è una costante dipendente dai dettagli del sistema simulato e dalla precisione richiesta alla simulazione. Contrariamente, un computer quantistico, utilizzerebbe kn qubits per la simulazione, dove k è sempre una costante dipendente dai dettagli del sistema. Ovviamente ogni simulazione è destinata a collassare in uno stato ben definito, di conseguenza la maggior parte dell'informazione dei c^n bit è **nascosta nella wave function** e non completamente accessibile ma, nonostante questo, la simulazione quantistica trova largo impiego in molti campi, soprattutto in quello della **Chimica quantistica**, per esempio per risolvere il problema della ricerca della **minima ground-state energy di una molecola** più o meno complessa; il problema citato, è affrontato dall'algoritmo ibrido "**variational quantum Eigensolver**" descritto in seguito.

Al momento la simulazione quantistica è l'unica classe di algoritmi utili in campo pratico e non puramente di ricerca, poiché richiede una precisione elevata ma non perfetta, mentre algoritmi come quello di Shor si basano su due **prerequisiti ad ora non realizzabili**, ovvero: la disponibilità di un **elevato numero di qubit**, e l'assunzione che siano **perfetti**, cioè non soggetti a errori.

Purtroppo al momento il computer quantistico con più elevato numero di qubit appartiene alla Google, e dispone di soli **54 qubit**, questo non basterebbe a fattorizzare, tramite l'algoritmo di Shor, interi a 3 cifre e inoltre riuscire a **isolare completamente un sistema quantistico** è un'enorme sfida alla tecnologia, che nonostante gli enormi progressi, è da considerare lontana dalla meta.

Per far fronte al **problema del rumore** nei computer quantistici si stanno sviluppando tecniche, dette di "**quantum error correction and dection**", analoghe a quelle usate nei classici protocolli di rete, ad esempio si possono utilizzare numerosi **qubit fisici**, di per sé imperfetti, per implementare alcuni **qubit logici** che si avvicinino alla perfezione, tanti più qubit reali sono impiegati tanto più bassa diventa la probabilità di errori, ma anche qui si ci ritrova a far fronte al problema dello **scarso numero di qubit** reali disponibili.

Di seguito affronteremo il già citato problema della **minima ground-state energy** di una molecola, passando per il problema, più semplice, del **taglio massimo**, il quale presenta **un'ottima analogia** poiché sfrutta anch'esso lo stesso algoritmo per essere risolto: il **VQE**.

4 Variational Quantum Eigensolver (VQE)

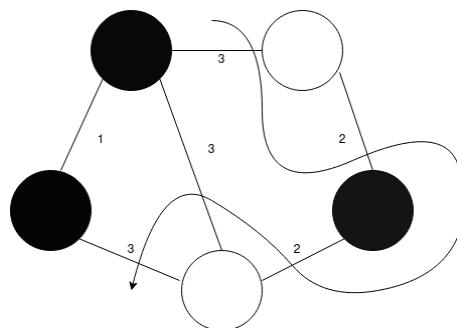
4.1 Introduzione al VQE: problema del taglio massimo (Max-Cut)

fonti bibliografiche: [3], [5].

Dato un Grafo $G(V,E)$ non diretto, con V insieme di n nodi $|V|=n$, e E degli archi, con pesi $w_{ij} > 0$, $w_{ij} = w_{ji} \forall ij \in V$: un **taglio** è una partizione dell'insieme dei nodi V in due sottoinsiemi e la **funzione costo** è la somma dei pesi degli archi che toccano nodi di sottinsiemi diversi. Il problema consiste nel trovare il taglio del grafo che massimizzi la funzione costo. Il problema è dimostrato essere NP-Completo, vedremo come tale problema sia risolvibile efficacemente utilizzando un algoritmo quantistico, il VQE. Assegnando un valore $x_i |x_i \in \{0, 1\}$ e $i \in V$, a seconda dell'insieme in cui si trova, il costo da ottimizzare è:

$$\tilde{C}(\mathbf{x}) = \sum_{i,j} w_{ij} x_i (1 - x_j)$$

Ad esempio nel seguente grafo il taglio è rappresentato dai due sottinsiemi di nodi di colore diverso, e il costo, dato dalla somma dei pesi sugli archi attraversati dalla curva, è 13



E possibile mappare la funzione costo su una matrice Hamiltoniana a partire dalla matrice di adiacenza di indici i, j , assegnando a $x_i \rightarrow \frac{(1-Z_i)}{2}$, dove Z_i è l'operatore Z di Pauli, ottendendo:

$$H = \frac{1}{2} \sum_{i < j} w_{ij} Z_i Z_j.$$

Ad esempio per la matrice di adiacenza del seguente grafo, l'hamiltoniana corrispondente è:

								<i>CBA</i>
1.5	0	0	0	0	0	0	0	$ 000\rangle$
0	-1.5	0	0	0	0	0	0	$ 001\rangle$
0	0	0.5	0	0	0	0	0	$ 010\rangle$
0	0	0	-0.5	0	0	0	0	$ 011\rangle$
0	0	0	0	-0.5	0	0	0	$ 100\rangle$
0	0	0	0	0.5	0	0	0	$ 101\rangle$
0	0	0	0	0	0	-1.5	0	$ 110\rangle$
0	0	0	0	0	0	0	1.5	$ 111\rangle$

4.1.1 Analogia con il problema della minima energia dello stato fondamentale

Lo stesso problema può essere formulato in termini di interazione magnetica tra elettroni: due elettroni vicini tendono ad avere spin opposto, ogni colore può rappresentare la direzione dello spin e il peso sugli archi la forza esercitata tra i due elettroni. La più bassa energia di un sistema E_G è quindi legata alla soluzione del Max-Cut.

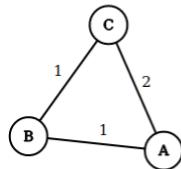
L'energia del sistema descritto dal taglio sul Grafo è data da

$$E = \frac{1}{2}(w_s - w_d)$$

Nell'equazione: w_s è la somma del peso degli archi tra nodi nello stesso sottoinsieme e w_d la somma del peso degli archi tra nodi in sottoinsiemi di versi.

Ogni possibile taglio ha una sua E , e possiamo mappare le varie energie su una matrice **Hamiltoniana**.

ad esempio nel seguente grafo ci sono 2^3 possibili tagli e la matrice H corrispondente è:



<i>CBA</i>	
2 0 0 0 0 0 0 0	000>
0 -1 0 0 0 0 0 0	001>
0 0 0 0 0 0 0 0	010>
0 0 0 -1 0 0 0 0	011>
0 0 0 0 -1 0 0 0	100>
0 0 0 0 0 0 0 0	101>
0 0 0 0 0 0 -1 0	110>
0 0 0 0 0 0 0 2	111>

A partire dalla matrice **H** ricavata è possibile risalire all'energia E di una qualsiasi configurazione moltiplicando H per il **vettore colonna** associato alla configurazione, e moltiplicando poi il **vettore riga** associato alla configurazione a quello risultante.

ad esempio per |011>:

$$H \begin{vmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{vmatrix} = \begin{vmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ -1 \\ 0 \end{vmatrix} \Rightarrow \begin{vmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{vmatrix} \begin{vmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ -1 \\ 0 \end{vmatrix} = -1$$

Dall'esempio si nota che il vettore associato alla configurazione è l'**Autovettore**, mentre l'Energia della configurazione è il relativo **Autovalore**, ne segue che la matrice H ha 8 possibili autovettori, le cui energie associate sono **Autovalori**.

In conclusione: è possibile, a partire da H, utilizzare il **VQE** per trovare l'autovettore con il **minimo autovalore** per H, che è analogo al problema della minima energia dello stato fondamentale in una molecola.

Infatti ciò che è stato fatto nell'esempio è riscrivibile come calcolo del **valore atteso** $\langle\psi|H|\psi\rangle$:

$$\langle 100 | H | 001 \rangle = -1$$

4.2 VQE nella simulazione di molecole

Una molecola formata da K nuclei e N elettroni, può essere interamente descritta dall'**Hermitiana molecolare H**

$$H = - \sum_i \frac{\hbar}{2m_e} \nabla_i^2 - \sum_I \frac{\hbar}{2M_I} \nabla_I^2 - \sum_{i,I} \frac{e^2}{4\pi\epsilon_0} \frac{Z_I}{|\mathbf{r}_i - \mathbf{R}_I|} + \\ + \frac{1}{2} \sum_{i \neq j} \frac{e^2}{4\pi\epsilon_0} \frac{1}{|\mathbf{r}_i - \mathbf{r}_j|} + \frac{1}{2} \sum_{I \neq J} \frac{e^2}{4\pi\epsilon_0} \frac{Z_I Z_J}{|\mathbf{R}_I - \mathbf{R}_J|}$$

dove M_i , R_i e Z_i rappresentano rispettivamente **massa**, **posizione** e **numero atomico** dell'i-esimo nucleo e r_i la **posizione** dell'i-esimo elettrone; I primi due termini indicano, rispettivamente, **l'energia cinetica di elettroni e nuclei**. Il terzo termine rappresenta **l'attrazione tra nuclei e elettroni**. Il quarto e quinto, rispettivamente, **la repulsione elettrone-elettrone e nucleo-nucleo**.

Il problema può essere ulteriormente ridotto grazie all'**approssimazione di Born-Oppenheimer** che, per la differenza di massa tra nuclei e elettroni, considera i primi non in movimento, rimuovendo così il secondo e l'ultimo termine. Come risultato si ottiene l'hamiltoniana elettronica:

$$H_e = - \sum_i \frac{\nabla_i^2}{2} - \sum_{i,I} \frac{Z_I}{|\mathbf{r}_i - \mathbf{R}_I|} + \frac{1}{2} \sum_{i \neq j} \frac{1}{|\mathbf{r}_i - \mathbf{r}_j|}$$

Il nostro obiettivo è l'approssimazione di una soluzione per l'**equazione di Schrödinger**:

$$H|\psi\rangle = E|\psi\rangle$$

In particolare siamo interessanti a trovare il minimo autovalore E_g detta **energia dello stato fondamentale**, associato alla matrice H che descrive il sistema, e per farlo sfruttiamo l'algoritmo **VQE**.

4.3 Algoritmo Variational Quantum Eigensolver

fonti bibliografiche: [2].

L'algoritmo **Variational Quantum Eigensolver**, è un algoritmo **ibrido**, ovvero sfrutta sia risorse quantistiche che classiche, inoltre risulta **robusto rispetto l'errore quantistico** e questo lo rende, potenzialmente, il primo algoritmo a poter superare i limiti della computazione classica, nonostante rimangano ancora sfide aperte riguardanti la parte di **ottimizzazione** eseguita su computer classici e la mitigazione del **rumore** quantistico.

Il VQE si basa sul **principio variazionale di Rayleigh-Ritz** il quale sancisce che, per una funzione d'onda parametrizzata $|\psi(\vec{\theta})\rangle$ e un'Hamiltoniana H che descrive il sistema, l' energia dello stato fondamentale E_g è t.c.

$$E_g \leq \langle \psi(\vec{\theta}) | H | \psi(\vec{\theta}) \rangle$$

Perciò, data H , con minimo autovalore λ_{min} , associato all'autostato $|\psi_{min}\rangle$, il VQE ci permette una stima di $\lambda_{\vec{\theta}}$ t.c.

$$\lambda_{min} \leq \lambda_{\vec{\theta}} \equiv \langle \psi(\vec{\theta}) | H | \psi(\vec{\theta}) \rangle$$

Dove $\vec{\theta}$ è un insieme di parametri e $\langle H \rangle(\vec{\theta}) \equiv \langle \psi(\vec{\theta}) | H | \psi(\vec{\theta}) \rangle$ è detto **Valore Atteso**. Questo implica che è possibile stimare E_g e $|\psi(\vec{\theta})\rangle$ trovando i giusti $\vec{\theta}$ che minimizzino il valore atteso.

Riassunto il VQE è descrivibile in tre fasi iterate ciclicamente fino a una convergenza:

1. il calcolo di $|\psi(\vec{\theta})\rangle$ tramite il vettore $\vec{\theta}$ e la var.form U ;
2. Misura del valore atteso $\langle H \rangle(\vec{\theta})$;
3. Determinare un nuovo range di parametri $\vec{\theta}_1$ t.c. $\langle H \rangle(\vec{\theta}_1) \leq \langle H \rangle(\vec{\theta})$ tramite un ottimizzatore classico

4.3.1 Forme Variazionali

Partendo da uno stato iniziale $|\psi\rangle$ detto **ansatz della funzione d'onda**, tramite un circuito parametrizzato fisso $U(\vec{\theta})$ detto **forma varazionale**, la **parte quantistica dell'algoritmo** fa variare l'ansatz in $|\psi(\vec{\theta})\rangle$, usata poi per il guessing di $\langle H \rangle(\vec{\theta})$.

L'azione di U è rappresentata dalla trasformazione lineare:

$$U(\vec{\theta})|\psi\rangle \equiv |\psi(\vec{\theta})\rangle$$

Successivamente la stima è migliorata tramite un **ottimizzatore classico** che modifica l'insieme di parametri θ al fine di **minimizzare il valore atteso** $\langle H \rangle(\vec{\theta})$.

È da notare che una variational form di n qubit, con un numero polinomiale di parametri, non può generare una qualsiasi trasformazione lineare dell'ansatz, questa la genererà solo in un sottospazio dello spazio di Hilbert; perciò l'abilità di avvicinarci al minimo autovalore λ_{min} è data solo da come l'ottimizzatore sceglie il vettore di parametri $\vec{\theta}$.

Le variational Form si suddividono in due categorie a seconda di come affrontano la precedente limitazione:

1. La prima categoria utilizza le conoscenze che si hanno **a priori** del dominio di applicazione per limitare i possibili output $|\psi(\vec{\theta})\rangle$ a un range utile, un esempio è la variational form **UCSSD**.
2. La seconda categoria invece sfrutta delle **euristiche** delle funzioni d'onda che ignorano del tutto le conoscenze a priori del dominio. Ne sono un esempio le variational form dette R_y , R_y e R_z che accettano alcune configurazioni come il **numero di qubit** del sistema, le impostazioni di **depth** e quelle di **entanglement**, dove la depth è il numero di ripetizioni del pattern di gates utilizzato, incrementando le ripetizioni possiamo **aumentare il numero di stati** $|\psi(\vec{\theta})\rangle$ ottenibili, ma al costo di utilizzare un **maggior numero di parametri** $\vec{\theta}$, mentre le impostazioni di entanglement specificano implicitamente il numero di CNOT gates della variational form, e possono essere di due tipi: **Linear e Full**.

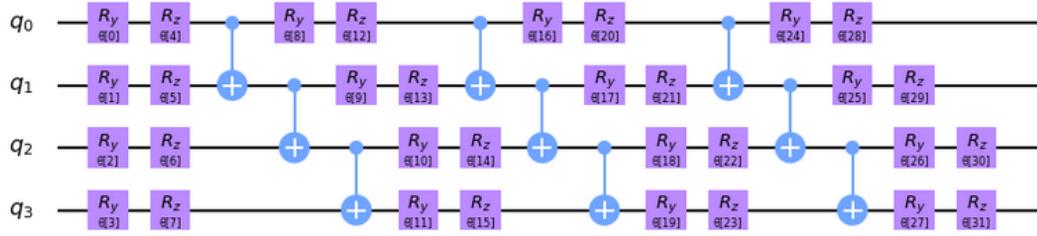
Ad esempio per $R_y R_z$ con 4 qubit:

```

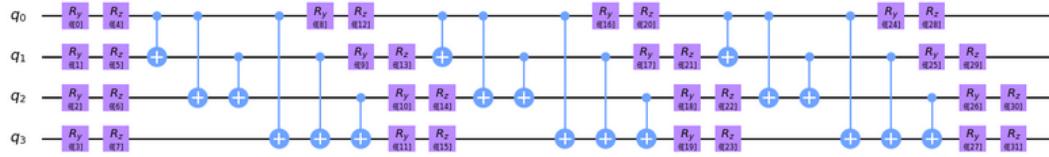
1 from qiskit.circuit.library import EfficientSU2
2 entanglements = ["linear", "full"]
3 for entanglement in entanglements:
4     form = EfficientSU2(num_qubits=4, entanglement=entanglement)
5     if entanglement == "linear":
6         print("Linear Entanglement:")
7     else:
8         print("Full Entanglement:")
9     display(form.draw(fold=100))
10    print()

```

Linear Entanglement:



Full Entanglement:



4.3.2 Ottimizzazione

Una volta scelta la variation form si procede scegliendo una strategia di ottimizzazione, al fine di scegliere l'insieme di parametri che minimizzino il valore atteso dell'Hamiltoniana target. Nonostante l'ottimizzazione venga eseguita sul un computer classico, anche qui , la scelta dell'algoritmo è legata al tipo di computer quantistico che userà i parametri, o del simulatore utilizzato.

Una strategia popolare di ottimizzazione, si basa sullo sfruttamento del **Gradiente**, dove ogni parametro è aggiornato nella direzione che porta al **maggior cambio locale di energia**. Questo metodo porta rapidamente a trovare un ottimo locale per il problema della minima energia dello stato fondamentale, con però l'evidente svantaggio di restare bloccato in una **soluzione parziale**.

Nel caso si utilizzi un vero hardware quantistico, o un simulatore rumoroso, un ottimizzatore classico approriato è l'**SPSA (Simultaneous Perturbation Stochastic Approximation)**. Questo, al contrario del precedente che modificava i parametri in base a caratteristiche individuali, **perturba i parametri casualmente**.

In una situazione ideale in cui il rumore non è presente, o in un simulatore noise-free (ad esempio lo *Statevector Simulator* in Qiskit Aqua), è disponibile una grande varietà di ottimizzatori, ne sono esempi : il **Sequential Least Squares Programming (SLSQP)** e il **Constrained Optimization by Linear Approximation (COBYLA)**.

5 VQE: diversi approcci verso una molecola di LiH in Qiskit

Nel seguente codice, utilizzando Qiskit, vengono mostrati vari approcci nell'utilizzo del VQE sull'Hamiltoniana di una molecola di LiH (idruro di litio) al fine di mettere in luce la differenza nell'uso di forme variazionali e ottimizzatori in contesti diversi.

```
1 from qiskit import BasicAer, Aer, IBMQ
2 from qiskit.aqua import QuantumInstance, aqua_globals
3 from qiskit.aqua.algorithms import VQE, ExactEigensolver
4 from qiskit.aqua.algorithms import NumPyEigensolver
5 from qiskit.aqua.components.initial_states import Zero
6 from qiskit.aqua.components.optimizers import COBYLA, L_BFGS_B, SLSQP, SPSA
7 from qiskit.aqua.components.variational_forms import RY, RYRZ, SwapRZ
8 from qiskit.aqua.operators import WeightedPauliOperator, Z2Symmetries
9 from qiskit.chemistry import FermionicOperator
10 from qiskit.chemistry.drivers import PySCFDriver, UnitsType
11 from qiskit.chemistry.components.variational_forms import UCCSD
12 from qiskit.chemistry.components.initial_states import HartreeFock
13 from qiskit.providers.aer import QasmSimulator
14 from qiskit.providers.aer.noise import NoiseModel
15 from qiskit.providers.aer.noise.errors import QuantumError, ReadoutError
16 from qiskit.providers.aer.noise.errors import pauli_error
17 from qiskit.providers.aer.noise.errors import depolarizing_error
18 from qiskit.providers.aer.noise.errors import thermal_relaxation_error
19 from qiskit.providers.aer import noise
20 from qiskit.ignis.mitigation.measurement import CompleteMeasFitter
21 provider = IBMQ.load_account()
22 import numpy as np
23 import matplotlib.pyplot as plt
24 from functools import partial
25 from qiskit.ignis.mitigation.measurement import CompleteMeasFitter
26 from qiskit.circuit.library import EfficientSU2
27 import warnings
28 warnings.filterwarnings('ignore', category=DeprecationWarning)
```

```
1 #Eigensolver classico per computare l'energia dello stato fondamentale
esatta, usata come valore di riferimento
2 def exact_solver(qubitOp):
3     ee = NumPyEigensolver(qubitOp)
4     result = ee.run()
5     ref = result['eigenvalues']
6     return ref
```

```
1 #Funzione usata per immagazzinare i risultati intermedi di una computazione
2 def store_intermediate_result(eval_count, parameters, mean, std):
3     counts.append(eval_count)
4     values.append(mean)
5     params.append(parameters)
6     deviation.append(std)
```

```

1 #Funzione di preparazione dell'Hamiltoniana su qubit a partire da una
2 #molecola di LiH
3 def compute_LiH_qubitOp(map_type, inter_dist, basis='sto3g'):
4     driver = PySCFDriver(atom='Li .0 .0 .0; H .0 .0 ' + str(inter_dist),
5     unit=UnitsType.ANGSTROM, charge=0, spin=0, basis=basis)
6
7     molecule = driver.run()
8     h1 = molecule.one_body_integrals
9     h2 = molecule.two_body_integrals
10    nuclear_repulsion_energy = molecule.nuclear_repulsion_energy
11
12    num_particles = molecule.num_alpha + molecule.num_beta
13    num_spin_orbitals = molecule.num_orbitals * 2
14
15    freeze_list = [0]
16    remove_list = [-3, -2]
17    remove_list = [x % molecule.num_orbitals for x in remove_list]
18    freeze_list = [x % molecule.num_orbitals for x in freeze_list]
19    remove_list = [x - len(freeze_list) for x in remove_list]
20    remove_list += [x + molecule.num_orbitals - len(freeze_list) for x in
21    remove_list]
22    freeze_list += [x + molecule.num_orbitals for x in freeze_list]
23
24    qubit_reduction = True if map_type == 'parity' else False
25
26    ferOp = FermionicOperator(h1=h1, h2=h2)
27    if len(freeze_list) > 0:
28        ferOp, energy_shift = ferOp.fermion_mode_freezing(freeze_list)
29        num_spin_orbitals -= len(freeze_list)
30        num_particles -= len(freeze_list)
31    if len(remove_list) > 0:
32        ferOp = ferOp.fermion_mode_elimination(remove_list)
33        num_spin_orbitals -= len(remove_list)
34
35    qubitOp = ferOp.mapping(map_type=map_type)
36    qubitOp = Z2Symmetries.two_qubit_reduction(qubitOp, num_particles) if
37    qubit_reduction else qubitOp
38    qubitOp.chop(10**-10)
39    total_en_shift = energy_shift + nuclear_repulsion_energy
40
41    return qubitOp, num_spin_orbitals, num_particles, qubit_reduction,
42    total_en_shift

```

```

1 #Definiamo un simulatore che consideri lo stato iniziale della simulazione e
2 #la distanza interatomica ma non il rumore quantistico,
3 backend = BasicAer.get_backend('statevector_simulator')
4 inter_dist = 1.5
5 qubitOp, num_spin_orbitals, num_particles, qubit_reduction, total_en_shift =
6 compute_LiH_qubitOp('parity',inter_dist)
7 initial_state = HartreeFock(num_spin_orbitals, num_particles,
8 qubit_mapping='parity')

```

```

1 #Calcoliamo l'energia esatta di riferimento tramite l'eigensolver classico
2 sopra descritto
3 ref = exact_solver(qubitOp)
4 exact_energy = np.real(ref[0] + total_en_shift)
5
6 print("Energia di riferimento: ",np.round(np.real(ref[0]),4))
7 print("Shift totale di energia: ",np.round(total_en_shift,4))
8 print("Energia esatta: ",np.round(exact_energy,4))

```

```

1 Energia di riferimento: -1.0991
2 Shift totale di energia: -6.782
3 Energia esatta: -7.881

```

5.1 Simulazioni senza Rumore

```

1 #Rumore: NO
2 #Ottimizzatore: SLSQP
3 #Variational form: UCCSD
4 optimizer = SLSQP(maxiter=10)
5 var_form = UCCSD(num_orbitals=num_spin_orbitals,
6 num_particles=num_particles, initial_state=initial_state,
7 qubit_mapping='parity')
8
9 counts=[]
10 values=[]
11 params=[]
12 deviation=[]
13
14 vqe = VQE(qubitOp, var_form, optimizer,callback=store_intermediate_result)
15 vqe_result_SLSQP = np.real(vqe.run(backend)['eigenvalue'] + total_en_shift)
16 print("Risultato del VQE con UCCSD e SLSQP:", np.round(vqe_result_SLSQP,5))
17 print("Energia di riferimento: ", np.round(exact_energy,5))
18
19 counts_SLSQP = counts
20 en_diff_SLSQP = values - np.real(ref[0])
21 dev_SLSQP = deviation

```

```

1 Risultato del VQE con UCCSD e SLSQP: -7.88102
2 Energia di riferimento: -7.88102

```

```

1 #Rumore: NO
2 #Ottimizzatore: SPSA
3 #Variational form: UCCSD
4 optimizer = SPSA(max_trials=200)
5
6 counts=[]
7 values=[]
8 params=[]
9 deviation=[]
10
11 vqe = VQE(qubitOp, var_form, optimizer,callback=store_intermediate_result)
12 vqe_result_SPSA = np.real(vqe.run(backend)['eigenvalue'] + total_en_shift)

```

```

13 print("Risultato del VQE con UCCSD e SPSA:", np.round(vqe_result_SPSA,5))
14 print("Energia di riferimento: ", np.round(exact_energy,5))
15
16 counts_SPSA = counts
17 en_diff_SPSA = values - np.real(ref[0])
18 deviation_SPSA = deviation

```

```

1 Risultato del VQE con UCCSD e SPSA: -7.75027
2 Energia di riferimento: -7.88102

```

```

1 #Rumore: NO
2 #Ottimizzatore: COBYLA
3 #Variational form: UCCSD
4 optimizer = COBYLA(maxiter=200, tol=0.001)
5
6 counts=[]
7 values=[]
8 params=[]
9 deviation=[]
10
11 vqe = VQE(qubitOp, var_form, optimizer,callback=store_intermediate_result)
12 vqe_result_COBYLA = np.real(vqe.run(backend)['eigenvalue'] + total_en_shift)
13 print("Risultato del VQE con UCCSD e COBYLA:",
14     np.round(vqe_result_COBYLA,5))
15 print("Energia di riferimento: ", np.round(exact_energy,5))
16
17 counts_COBYLA = counts
18 en_diff_COBYLA = values - np.real(ref[0])
19 deviation_COBYLA = deviation

```

```

1 Risultato del VQE con UCCSD e COBYLA: -7.88101
2 Energia di riferimento: -7.88102

```

```

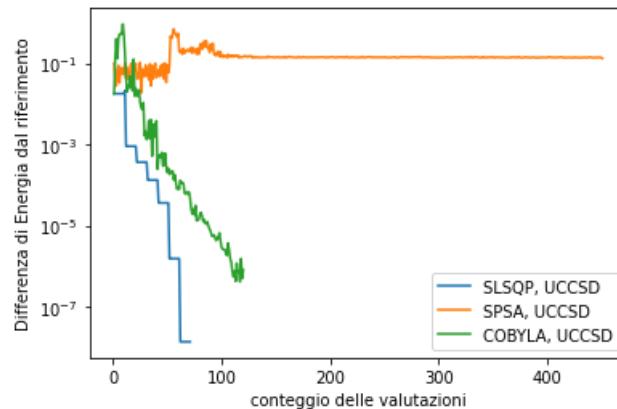
1 #Riportiamo su un grafico l'andamento delle varie simulazioni
2 plt.plot(counts_SLSQP,en_diff_SLSQP,label="SLSQP, UCCSD")
3 plt.plot(counts_SPSA,en_diff_SPSA,label="SPSA, UCCSD")
4 plt.plot(counts_COBYLA,en_diff_COBYLA,label="COBYLA, UCCSD")
5 plt.xlabel('conteggio delle valutazioni')
6 plt.ylabel('Differenza di Energia dal riferimento')
7 plt.yscale("log")
8 plt.legend()
9
10 print("Energia di riferimento: ", np.round(exact_energy,5))
11 print("Risultato del VQE con UCCSD e SLSQP:", np.round(vqe_result_SLSQP,5))
12 print("Risultato del VQE con UCCSD e SPSA:", np.round(vqe_result_SPSA,5))
13 print("Risultato del VQE con UCCSD e COBYLA:",
14     np.round(vqe_result_COBYLA,5))
15

```

```

1 | Energia di riferimento: -7.88102
2 | Risultato del VQE con UCCSD e SLSQP: -7.88102
3 | Risultato del VQE con UCCSD e SPSA: -7.75027
4 | Risultato del VQE con UCCSD e COBYLA: -7.88101

```



L'immagine illustra le differenze nell'esecuzione dell'algoritmo con variational form UCCSD fissata e ottimizzatori SLSQP, SPSA e COBYLA, rispettivamente indicati dai colori blu, arancione, verde

```

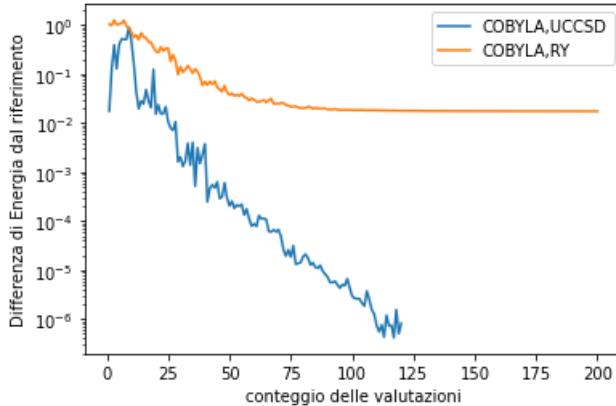
1 #Rumore: NO
2 #Ottimizzatore: SPSA
3 #Variational form: RY
4 RY_var_form = RY(qubitOp.num_qubits, depth=1)
5
6 var_form=RY_var_form
7 optimizer = COBYLA(maxiter=200, tol=0.001)
8
9 counts=[]
10 values=[]
11 params=[]
12 deviation=[]
13
14 vqe = VQE(qubitOp, var_form, optimizer, callback=store_intermediate_result)
15 result_RY = np.real(vqe.run(backend)[ 'eigenvalue' ] + total_en_shift)
16 print("Risultato del VQE con RY e COBYLA:::", np.round(result_RY,5))
17 print("Energia di riferimento: ", np.round(exact_energy,5))
18
19 counts_RY_COBYLA = counts
20 en_diff_RY_COBYLA = values - np.real(ref[0])
21 deviation_RY_COBYLA = deviation
22
23 #Osserviamo graficamente la differenza tra le due variational forms
24 plt.plot(counts_COBYLA,en_diff_COBYLA,label="COBYLA,UCCSD")
25 plt.plot(counts_RY_COBYLA,en_diff_RY_COBYLA,label="COBYLA,RY")
26 plt.xlabel('conteggio delle valutazioni')
27 plt.ylabel('Differenza di Energia dal riferimento')
28 plt.yscale("log")
29 plt.legend()

```

```

1 | Risultato del VQE con RY e COBYLA:: -7.86354
2 | Energia di riferimento: -7.88102

```



L'immagine illustra le differenze nell'esecuzione dell'algoritmo con ottimizzatore fissato COBYLA e variational form UCSSD e RY, rispettivamente indicati dai colori blu e arancione

5.2 Simulazioni in presenza di Rumore

Introduciamo ora il rumore, definendo un **simulatore rumoroso** il quale utilizza un modello di rumore basato sul chip dell'IBM **ibmq_essex**; ricordiamo inoltre che l'**SPSA è l'ottimizzatore più efficace in presenza di rumore**. Nelle due simulazioni successive, osservando i grafici in output, si può notare che, nello stesso scenario rumoroso, l'SPSA si dimostri un approccio migliore rispetto al COBYLA.

```

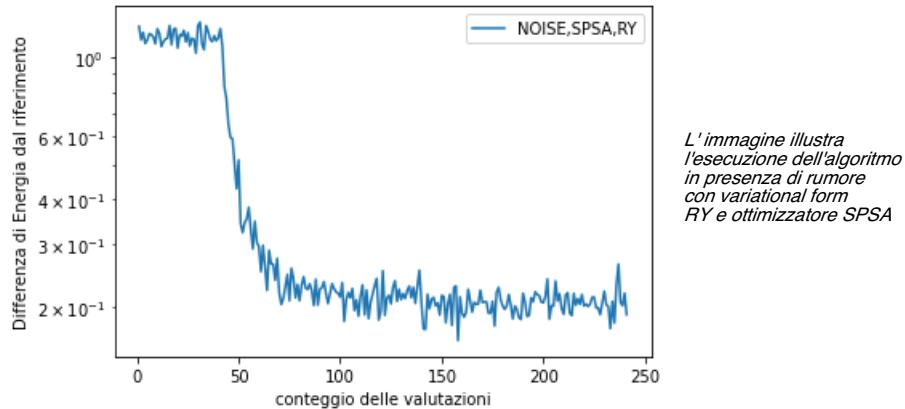
1 #Rumore: SI
2 #Ottimizzatore: SPSA
3 #Variational form: RY
4 chip_name = 'ibmq_essex'
5 device = provider.get_backend(chip_name)
6 coupling_map = device.configuration().coupling_map
7 noise_model = NoiseModel.from_backend(device.properties())
8 simulator = Aer.get_backend('qasm_simulator')
9
10 quantum_instance = QuantumInstance(backend=simulator,
11                                     shots=1000,
12                                     noise_model=noise_model,
13                                     coupling_map=coupling_map,
14
15                                     measurement_error_mitigation_cls=CompleteMeasFitter,
16                                     cals_matrix_refresh_period=30)
17
18 var_form = RY(num_qubits=4, depth=1)
19 optimizer = SPSA(max_trials=100)
20
21 counts=[]
22 values=[]
23 params=[]
24 deviation=[]
25
26 vqe = VQE(qubitOp, var_form,
27             optimizer=optimizer, callback=store_intermediate_result)
28 vqe_result_RY_SPSA_d1 = np.real(vqe.run(quantum_instance)['eigenvalue'])
29
30 print("Risultati del VQE con rumore, variational form 'RY' e ottimizzatore 'SPSA'")
31 print("Energia stimata dal VQE: ", np.round(vqe_result_RY_SPSA_d1+total_en_shift,5))
32 print("Energia di riferimento: ", np.round(exact_energy,5))
33 print("Differenza con il riferimento: ", np.round(vqe_result_RY_SPSA_d1-np.real(ref[0]),4))
```

```

32
33 counts_noise_SPSA_RY = counts
34 en_diff_noise_SPSA_RY = values - np.real(ref[0])
35 deviation_noise_SPSA_RY = deviation
36
37 plt.plot(counts_noise_SPSA_RY,en_diff_noise_SPSA_RY,label="NOISE,SPSA,RY")
38 plt.xlabel('conteggio delle valutazioni')
39 plt.ylabel('Differenza di Energia dal riferimento')
40 plt.yscale("log")
41 plt.legend()

```

1 Risultati del VQE con rumore, variational form 'RY' e ottimizzatore 'SPSA'
2 Energia stimata dal VQE: -7.69125
3 Energia di riferimento: -7.88102
4 Differenza con il riferimento: 0.1898



```

1 #Rumore: SI
2 #Ottimizzatore: COBYLA
3 #Variational form: RY
4 chip_name = 'ibmq_essex'
5 device = provider.get_backend(chip_name)
6 coupling_map = device.configuration().coupling_map
7 noise_model = NoiseModel.from_backend(device.properties())
8 simulator = Aer.get_backend('qasm_simulator')

9
10 quantum_instance = QuantumInstance(backend=simulator,
11                                     shots=1000,
12                                     noise_model=noise_model,
13                                     coupling_map=coupling_map,
14
15                                     measurement_error_mitigation_cls=CompleteMeasFitter,
16                                     cals_matrix_refresh_period=30)

17 var_form = RY(num_qubits=4, depth=1)
18 optimizer = COBYLA(maxiter=200, tol=0.001)
19
20 counts=[]
21 values=[]
22 params=[]
23 deviation=[]
24
25 vqe = VQE(qubitOp, var_form,
26            optimizer=optimizer,callback=store_intermediate_result)
27 vqe_result_RY_COBYLA_d1 = np.real(vqe.run(quantum_instance)[eigenvalue])

```

```

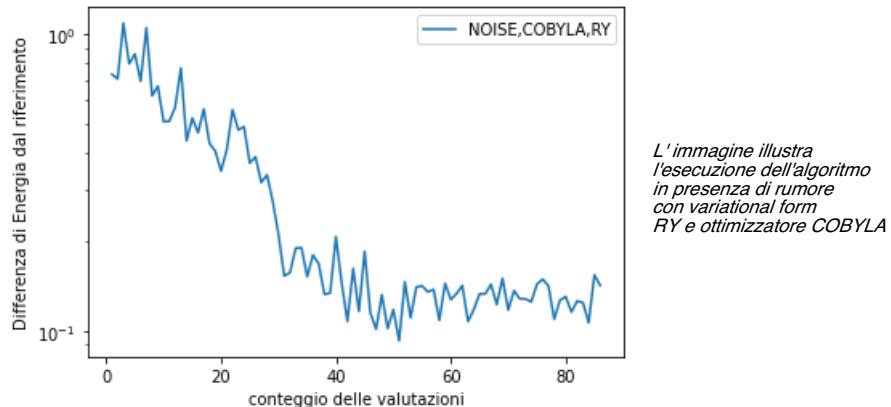
27
28 print("Risultati del VQE con rumore, variational form 'RY' e ottimizzatore
29 'SPSA'")
30 print("Energia stimata dal VQE:
31     ", np.round(vqe_result_RY_COBYLA_d1+total_en_shift,5))
32 print("Energia di riferimento: ", np.round(exact_energy,5))
33 print("Differenza con il riferimento: ", np.round(vqe_result_RY_COBYLA_d1-
34     np.real(ref[0]),4))
35
36
37 counts_noise_COBYLA_RY = counts
38 en_diff_noise_COBYLA_RY = values - np.real(ref[0])
39 deviation_noise_COBYLA_RY = deviation
40
41 plt.plot(counts_noise_COBYLA_RY,en_diff_noise_COBYLA_RY,label="NOISE, COBYLA,
42 RY")
43 plt.xlabel('conteggio delle valutazioni')
44 plt.ylabel('Differenza di Energia dal riferimento')
45 plt.yscale("log")
46 plt.legend()

```

```

1 Risultati del VQE con rumore, variational form 'RY' e ottimizzatore 'SPSA'
2 Energia stimata dal VQE: -7.73843
3 Energia di riferimento: -7.88102
4 Differenza con il riferimento: 0.1426

```



Bibliografia

- [1] M.A. Nielsen and I.L. Chuang. *Quantum Computation and Quantum Information*. Cambridge Series on Information and the Natural Sciences. Cambridge University Press, 2000. isbn: 9780521635035.
- [2] A. Asfaw and L. Bello and Y. Ben-Haim and S. Bravyi and N. Bronn and L. Capelluto and A.C. Vazquez and J. Ceroni and R. Chen and A. Frisch and J. Gambetta and S. Garion and L. Gil and S.D.L.P. Gonzalez and F. Harkins and T. Imamichi and D. McKay and A. Mezzacapo and Z. Minev and R. Movassagh and G. Nannicini and P. Nation and A. Phan and M. Pistoia and A. Rattew and J. Schaefer and J. Shabani and J. Smolin and K. Temme and M. Tod and S. Wood and J. Wootton. *Learn Quantum Computation Using Qiskit*. 2020. URL: <http://community.qiskit.org/textbook>.
- [3] Giacomo Nannicini. *An Introduction to Quantum Computing, Without the Physics*. IBM T.J. Watson, Yorktown Heights, NY, 2020. URL: <https://arxiv.org/pdf/1708.03684.pdf>.
- [4] James Weaver. *The Variational Quantum Eigensolver. Examining the inner-workings of the VQE algorithm*. 2019. URL: <https://medium.com/qiskit/the-variational-quantum-eigensolver-43f7718c2747>.
- [5] A. Mezzacapo, J. Gambetta, K. Temme, R. Movassagh, A. Frisch, T. Imamichi, G. Nannicini, R. Chen, M. Pistoia, S. Wood. *Qiskit Aqua: Experimenting with MaxCut problem and Traveling Salesman problem with variational quantum eigensolver*. 2020. URL: https://github.com/Qiskit/qiskit-tutorials/blob/master/tutorials/optimization/6_examples_max_cut_and_tsp.ipynb.

Conclusione e prospettive future

Sono stati introdotti i concetti chiave della computazione quantistica, i qubit, la sovrapposizione, i gate quantistici e come implementare dei circuiti quantistici per sfruttarne le proprietà; è stato analizzato il potere computazionale di questo paradigma e dato uno sguardo alle applicazioni che può avere, soffermandoci su algoritmi quali la trasformata quantistica di Fourier, la stima della fase quantistica, l'algoritmo di Shor e il variational quantum eigensolver, con le sue varianti, di cui alcune ne è stata implementata in python una reale applicazione per la molecola di LiH.

Si è cercato di far risaltare l'importanza che attualmente ricopre la simulazione quantistica, poiché date le native proprietà dei sistemi quantistici, nel peggio dei casi, ci aspettiamo che almeno nel campo della simulazione i computer quantistici siano estremamente più performanti di quelli classici. È stato trattato l'attuale stato della computazione quantistica, ma cosa si può dire riguardo al futuro? Cosa ha da offrire la computazione quantistica alla scienza, alla tecnologia e all'umanità? Quali progressi fornisce alle discipline da cui è stata generata quali l'informatica, la teoria dell'informazione, la fisica e la chimica?

Abbiamo visto che le applicazioni a lungo termine per i computer quantistici sono molte ed estremamente promettenti, che già quelle attuali portano benefici in svariati campi, e che restano ancora molti problemi aperti, per citarne alcuni: la teoria del rilevamento e della correzione degli errori con l'elevata quantità di qubit fisici di cui ha bisogno per essere efficiente, capire quali problemi rientrano nella classe computazionale BPQ, come riuscire a isolare singoli qubit nei processori quantistici e molti ancora. Inoltre restano molti problemi riguardanti la fisica delle particelle che non siamo in grado di risolvere, e questi (come la teoria della gravità quantistica) sono di natura quantomeccanica, quindi il modo migliore per affrontarli computazionalmente, è avere un dispositivo che incorpori nativamente queste proprietà, e sia quindi in grado di simulare efficacemente tali sistemi. Questa tesi offre solo uno sguardo all'enorme quantità di aree di ricerca e di applicazioni che si diramano da questa disciplina, spero però sia sufficiente a dare un'intuizione sul modo di operare degli algoritmi quantistici, e sulle loro intrinseche potenzialità.