POLITECNICO DI TORINO

Ingegneria Informatica

Laurea magistrale in Data Science

Master Degree Thesis

Big Data processing and Quantum Computing

Supervisor
Prof. Montrucchio Bartolomeo

Co-Supervisor
Prof. Filippo Gandino
Ph.D. Student Edoardo Giusto
Ph.D. Student Mohammad Ghazi Vakili

Internship Tutor
Andrea Bardone
Giuseppe Cotugno

Candidato
Alessandra MUSONE
Matricola: s255653

Academic Year 2019 - 2020

Sommario

Le tecnologie Big Data offrono soluzioni innovative e nuove architetture in grado di gestire quantità sempre più crescenti di dati di varia natura, usando intensamente il calcolo distribuito. Parallelamente, il quantum computing consente un incremento esponenziale delle prestazioni di calcolo e la risoluzione di problemi complessi. Queste nuove tecnologie abilitano l'implementazione di algoritmi per la soluzione di problemi di ottimizzazione in ambito industriale, logistico e medico. L'obiettivo della tesi è la definizione, la valutazione e l'implementazione di un algoritmo di ottimizzazione su un caso d'uso in ambito logistico tramite Quantum Computing (IBM, D.Wave).

Indice

Ai miei genitori

Introduzione

Ogni giorno vengono creati più di 2,5 Exabyte di dati.
Questo numero è in continua crescita grazie all'ascesa del 5G e dell'Internet of Things (IoT).
L'apprendimento automatico (Machine Learning) e l'intelligenza artificiale sono due modi per analizzare e gestire i dati, tuttavia il desiderio di approfondimenti e la continua innovazione rendono il processo di raccolta e analisi dei dati sempre più complesso.
Le tecnologie e gli strumenti offerti da Big Data propongono soluzioni innovative e nuove strutture in grado di gestire un volume sempre più crescente di dati.
Parallelamente, il computer quantistico incrementa esponenzialmente le prestazioni di calcolo, calcolando la risoluzione di problemi complessi in tempi estremamente ridotti.
Oggigiorno man mano che l'informatica classica giunge ai suoi limiti prestazionali, si afferma sempre più l'informatica quantistica che risulta essere una delle tendenze digitali in più rapida crescita e si pensa possa essere la soluzione per le future sfide dei big data.
I computer quantistici permettono l'implementazione di algoritmi efficienti soprattutto per la risoluzione di problemi di ottimizzazione in ambito logistico, industriale e medico. L'obiettivo della tesi è la definizione, la valutazione e l'implementazione di un algoritmo di ottimizzazione su un caso d'uso in ambito logistico tramite Quantum Computing.

Obiettivi

1. Approfondire il funzionamento del computer quantistico.
2. Capire gli ambiti applicativi di un computer quantistico e i suoi vantaggi.
3. Motivare la scelta del compuer e della libreria utilizzata.
4. Introdurre i problemi di ottimizzazione, definire un caso d'uso e risolverlo attraverso un computer quantistico.
5. Analizzare le soluzione trovata e i limiti hardware riscontrati.

Struttura della tesi

La tesi è suddivisa in quattro sezioni in modo da fornire una lettura strutturata dell'elaborato

1. Teoria del Quantum Computing e stato dell'arte.
2. Applicazioni del quantum .

3. Problema di ottimizzazione preso in esame e soluzione proposta.
4. Confronto delle soluzioni trovate e conclusione.

Capitolo 1

Il computer quantistico

"È assurdo impiegare gli uomini di intelligenza eccellente per fare calcoli che potrebbero essere affidati a chiunque se si usassero delle macchine."

— Leibnitz

1.1 Teoria della computazione

1.1.1 Introduzione alla computazione classica, la macchina di Turing

La macchina di Turing è una macchina ideale che manipola i dati contenuti su un nastro di lunghezza potenzialmente infinita, secondo un insieme prefissato di regole ben definite. Ad ogni istante la macchina si trova in uno stato appartenente ad un insieme finito e legge un carattere sul nastro. La funzione di transizione, in modo deterministico, specifica tre cose: il simbolo da scrivere sul nastro, la direzione in cui la testina deve muoversi, e il successivo stato della macchina.

Nella macchina di Turing deterministica (TM) l'insieme di regole prevede al massimo un'azione da eseguire per ogni situazione. Al contrario, nella macchina di Turing non deterministica (NTM) l'insieme di regole possono prevedere più azioni da eseguire per ogni situazione.
Una computazione eseguita da una macchina non deterministica può essere rappresentata con un albero di computazioni deterministiche.

La computazione classica si basa sul modello della macchina di Turing, che risulta essere un modello astratto di macchina universale.

La macchina di Turing funziona seguendo un insieme di regole e di principi enunciati da Alan Turing nel 1936 [1] e successivamente elaborati da John von Neumann negli anni '40. Alla base di questi principi vi è l'assunzione che la macchina di Turing idealizza un dispositivo che obbedisce alle regole della fisica classica.

1.1.2 La computazione quantistica

La computazione quantistica, a differenza della computazione classica, nasce come paradigma alternativo. Si basa sui principi della Meccanica quantistica, che sono gli unici in grado di giustificare i fenomeni fisici a livello microscopico, come ad esempio all'interno di un atomo.
Quindi la Meccanica quantistica spiega i fenomeni che avvengono a livello atomico per i quali la Meccanica classica (o newtoniana) è del tutto inadeguata.

La caratteristica fondamentale della Meccanica quantistica è che descrive la materia e la radiazione sia come entità particellare che come fenomeno ondulatorio. Al contrario la Meccanica classica non gode del dualismo onda-particella, infatti la luce è descritta come onda e l'elettrone come una particella.

Con la computazione quantistica viene introdotta la teoria della quantizzazione. A livello microscopico, le quantità fisiche come per ad esempio l'energia non sono scambiabili in modo "continuo", ma attraverso "pacchetti" detti quanti. In base a questa proprietà, la luce è costituita da corpuscoli di energia chiamati "fotoni".

Le prime applicazioni di questo metodo di calcolo non convenzionale si ebbero agli inizi degli anni Ottanta. In quegli anni, infatti, cominciò ad affacciarsi l'idea di realizzare un modello di computazione come un sistema quantistico allor quando il fisico statunitense P.Benioff pubblicò una descrizione completa del modello meccanico quantistico della Macchina di Turing [2]. Il modello computazionale era basato su considerazioni precedentemente elaborate da C.Bennet che definiscono la Macchina di Turing reversibile [3].
Una computazione è reversibile quando il processo di calcolo è invertibile. Di conseguenza, si ritorna allo stato inziale ripercorrendo all'indietro i vari passi eseguiti. Benioff è stato il primo a dimostrare che il calcolo quantistico reversibile è teoricamente possibile. La sua ricerca aprì la strada a numerosi altri scienziati che approfondirono il campo dell'informatica quantistica.
R. Feynman, successivamente, dimostrò che nessuna Macchina di Turing classica riusciva a simulare alcuni fenomeni fisici senza subire un rallentamento esponenziale delle prestazioni. Un "simulatore quantistico universale", al contrario, avrebbe potuto effettuare la simulazione in maniera molto più efficiente [4].
Nel 1985 fu fondamentale il contributo del fisico britannico D. Deutsch [5]. Egli formalizzò le idee fino ad allora formulate nella sua Macchina di Turing Quantistica Universale, che ha portato alla concezione moderna di computazione quantistica. Basandosi sulla tesi di Church-Turing, David Deutsch si spinse inoltre ad avanzare il principio secondo cui "ciascun sistema fisico realizzabile può essere perfettamente simulato da un modello universale di macchina computazionale operante con risorse finite". Secondo Deutsch, questo principio implica che un computer quantistico è in grado di simulare qualsiasi esperimento fisico .

L'introduzione del nuovo modello di calcolo ha influenzato significativamente il

campo della complessità computazionale, provocando il cambiamento della nozione di "trattabilità". Infatti, nel 1994 P. Shor dimostra che il problema della fattorizzazione dei numeri primi, che fino ad allora era stato considerato intrattabile, si può risolvere efficientemente, in tempo polinomiale, con un algoritmo quantistico [6]. Queste considerazioni, unite a quelle di tipo tecnologico, hanno portato all'affermarsi di un campo di ricerca del tutto nuovo e allo sviluppo di computer in grado di risolvere problemi complessi e lunghi in tempi brevissimi. In particolare nei prossimi capitoli si affronteranno tre fenomeni fondamentali della teoria quantistica su cui la computazione quantistica si basa in maniera essenziale, e che determinano la sua enorme potenzialità di calcolo: il principio di sovrapposizione degli stati, il principio di misurazione e il fenomeno dell'entanglement.

1.1.3 Teoria della complessità

La teoria della complessità è lo studio dello sforzo computazionale richiesto per eseguire un algoritmo. Considerando un qualsiasi algoritmo per risolvere un dato problema, si può studiare lo sforzo computazionale inerente alla ricerca della soluzione.

La moltiplicazione, ad esempio, non è un'operazione così semplice. Gli algoritmi insegnati nelle scuole per moltiplicare due numeri richiedono $O(n^2)$ operazioni base, come addizioni e moltiplicazioni a una cifra. Sebbene siano stati trovati algoritmi con una complessità asintotica inferiore, è comunque considerato impossibile eseguire la moltiplicazione con complessità di $O(n)$.

La moltiplicazione in fin dei conti però non è un problema complesso. Un esempio di problema con una complessità molto maggiore è la fattorizzazione. Si considera un numero di n cifre di cui bisogna trovare i fattori primi. L'algoritmo più noto in questo caso ha una complessità di $O(e^{(n^{(1/3)})})$. L'esponenziale qui significa che la complessità cresce molto rapidamente e rende la fattorizzazione un problema molto difficile da risolvere.

Considerando un numero con molte cifre, ad esempio 829 cifre, eseguire la fattorizzazione su questo numero richiede un super-computer e circa 2700 anni-core. Per la fattorizzazione di numeri più grandi, si arriva facilmente a un punto in cui un super-computer delle dimensioni di un pianeta dovrebbe funzionare per l'età dell'universo. Qualsiasi problema del genere è praticamente impossibile.
Attraverso l'utilizzo di un computer quantistico, la complessità si riduce di molto.

1.1.4 Classi di complessità

Si definisce classe di complessità un insieme di problemi di una determinata complessità [7].

Eseguendo i problemi su un modello computazionale astratto, solitamente una macchina di Turing, si individua la particolare classe di complessità in cui rientrano.

Le due classi più importanti sono P e NP. La classe P è l'insieme dei problemi di decisione che possono essere risolti da una macchina di Turing deterministica in tempo polinomiale. In P ci sono tutti i problemi che un computer classico può risolvere rapidamente, ad esempio: "Questo numero è primo?".

La classe NP è l'insieme dei problemi di decisione che possono essere risolti da una macchina di Turing non deterministica in tempo polinomiale. In NP ci sono tutti i problemi che i computer classici non riescono necessariamente a risolvere rapidamente, ad esempio: "Quali sono i suoi fattori primi?".

Inoltre, all'interno della classe NP, vi sono il problema NP-Complete e NP-hard.

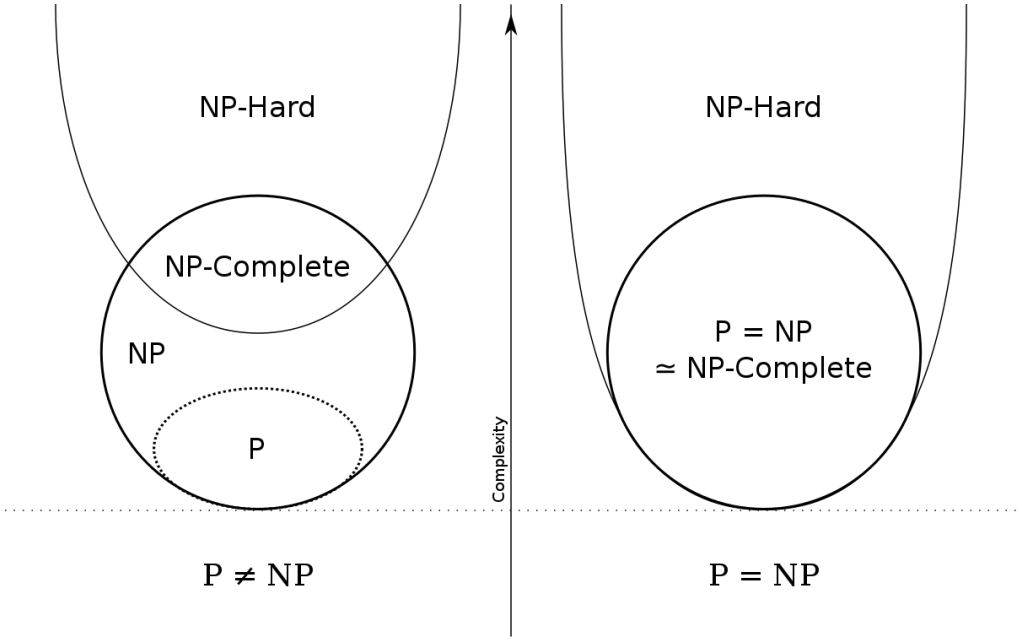


Figura 1.1. Le classi di complessità per P=NP e P != NP. Immagine tratta da Wikipedia

NP-Complete è una classe di complessità che rappresenta l'insieme di tutti i problemi X in NP per i quali è possibile ridurre qualsiasi altro problema NP Y a X in tempo polinomiale. Intuitivamente questo significa che si può risolvere rapidamente Y se si sa come risolvere rapidamente X.

Un problema X è NP-hard, invece, se c'è un problema NP-Complete Y, tale che Y è riducibile a X in tempo polinomiale. Intuitivamente, questi sono i problemi che sono difficili almeno quanto i problemi NP-Complete. Si fa notare che i problemi NP-hard non devono essere necessariamente in NP e non devono essere problemi decisionali. Se c'è una soluzione a un problema NP-hard in tempo polinomiale, c'è una soluzione a tutti i problemi NP in tempo polinomiale.

Nel 1993 gli informatici Ethan Bernstein e Umesh Vazirani definirono una nuova classe di complessità chiamata BQP [8] (per Bounded-error Quantum Polynomial

time): la classe di complessità dei problemi decisionali che possono essere risolti con un errore bilaterale su una macchina di Turing quantistica in tempo polinomiale. Questa particolare classe contiene tutti i problemi decisionali che i computer quantistici possono risolvere in modo efficiente.
Ethan Bernstein e Umesh Vazirani hanno anche dimostrato che i computer quantistici possono risolvere tutti i problemi che i computer classici possono risolvere: BQP contiene tutti i problemi che si trovano in P.
Un compito centrale della teoria del calcolo quantistico è capire come il BQP si adatta alle classi di complessità classiche.
Di seguito il grafico descrittivo delle classi di complessità, i cui confini però della classe BQP sono incerti.

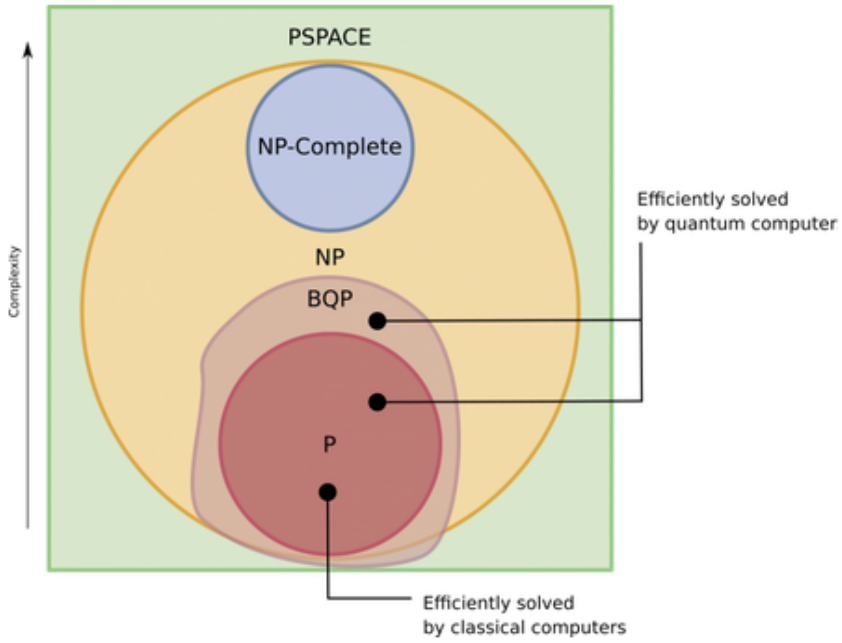


Figura 1.2. BQP e come si adatta alle classi di complessità classiche, Immagine tratta da www.quantum-bits.org

1.2 Il computer Classico

La programmazione di un computer quantistico, al giorno d'oggi, può svolgersi anche tra le mura domestiche.

Che cos'è, dunque, un programma quantistico? E come funziona nel dettaglio un computer quantistico?

E' possibile rispondere a queste domande facendo confronti con computer digitali standard.

I computer classici funzionano a bit. I bit hanno solo due valori, 0 e 1, con i quali si può rappresentare qualsiasi informazione.
Un numero è rappresentato da dieci cifre 0, 1, 2, 3, 4, 5, 6, 7, 8 e 9, ognuna delle quali, in base alla posizione che occupa, assume un valore diverso.
Ad esempio, quando scriviamo 9213, intendiamo: 9000+200+10+3, oppure espresso in modo che visualizzi le potenze del 10 : $(9\times10^3)+(2\times10^2)+(1\times10^1)+(3\times10^0)$.
Sebbene di solito viene utilizzato un sistema decimale, quindi basato sul numero 10, è possibile usarne uno basato su qualsiasi altro numero. Il sistema numerico binario, ad esempio, si basa sul numero due. Ciò significa utilizzare i due caratteri 0 e 1 per esprimere i numeri come multipli di potenze di due. Ad esempio, 9213 diventa 10001111111101, quindi: $(1\times2^{13})+(0\times2^{12})+(0\times2^{11})+(0\times2^{10})+(1\times2^9)+(1\times2^8)+(1\times2^7)+(1\times2^6)+(1\times2^5)+(1\times2^4)+(1\times2^3)+(1\times2^2)+(0\times2^1)+(1\times2^0)$.
Attraverso i bit è possibile rappresentare oltre che numeri qualsiasi lettera, carattere speciale o segno di punteggiatura che si desidera utilizzare.
Questo è il modo in cui tutte le informazioni vengono rappresentate nei computer. Che si tratti di numeri, lettere, immagini o suoni, tutto esiste sotto forma di bit.

1.3 Introduzione al computer quantistico

1.3.1 Una panoramica generale ai computer del futuro

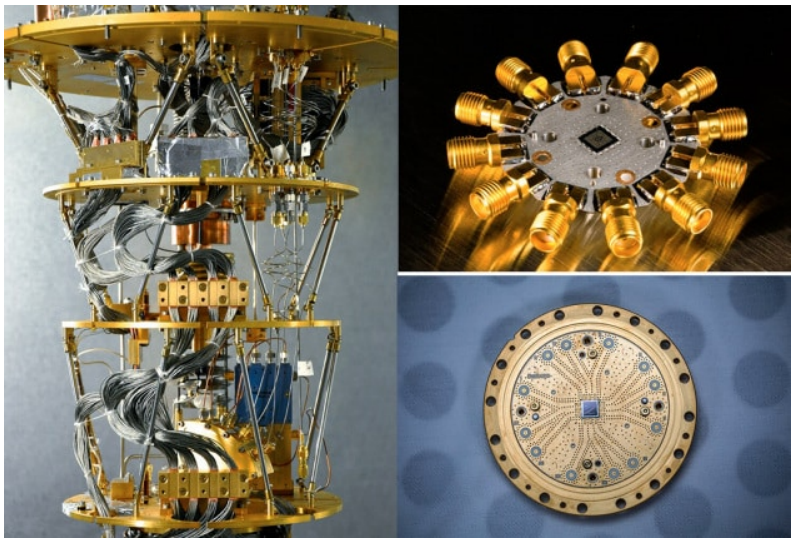


Figura 1.3. Hardware di un computer quantistico. Immagine tratta da www.focus.it

I computer quantistici sono macchine che utilizzano le proprietà della fisica quantistica per memorizzare dati ed eseguire calcoli. Questo può essere estremamente vantaggioso per svolgere alcuni compiti in cui le prestazioni potrebbero ampiamente superare quelle dei migliori super-computer presenti attualmente.

I computer attuali manipolano singoli bit, che memorizzano le informazioni come stati binari, 0 e 1. I computer quantistici, invece, si basano su bit quantici o qubit.
I Qubit sono realizzati usando sistemi fisici, come lo spin di un elettrone o l'orientamento di un fotone.
Questi sistemi possono trovarsi contemporaneamente in stati diversi, grazie a una proprietà nota come sovrapposizione quantistica.
I Qubit, inoltre, possono essere indissolubilmente legati tra loro, tale fenomeno è chiamato entanglement quantistico. Il risultato è che una serie di qubit può rappresentare simultaneamente tanti stati diversi.

Ad esempio, otto bit sono sufficienti per un computer classico per rappresentare qualsiasi numero compreso tra 0 e 255. Otto qubit sono sufficienti per un computer quantistico per rappresentare otto numeri compresi tra 0 e 255 contemporaneamente.
Poche centinaia di qubit intrecciati sarebbero sufficienti per rappresentare più numeri di quanti sono gli atomi nell'universo.

Sfruttando la capacità unica delle particelle subatomiche che consente loro di esistere in più di uno stato, cioè 1 e 0 contemporaneamente, la sovrapposizione e l'entanglement sono due caratteristiche della fisica quantistica su cui si basano i computer quantistici. Ciò consente di gestire le operazioni a velocità esponenzialmente superiori rispetto ai computer convenzionali e con un consumo di energia molto inferiore.

In situazioni in cui esiste un gran numero di possibili combinazioni, i computer quantistici possono considerarle tutte contemporaneamente. Tuttavia, ci possono essere anche molte situazioni in cui i computer classici risultano migliori rispetto a quelli quantistici, di conseguenza i computer del futuro potrebbero essere pensati come una combinazione tra computer classico e computer quantistico.

Attualmente i computer quantistici sono estremamente sensibili: il calore, i campi elettromagnetici e le collisioni con molecole d'aria possono far perdere le proprietà quantiche a un qubit. Questo processo, noto come decoerenza quantistica, è tanto più rapido quante più particelle sono coinvolte e provoca il crash del sistema.
Inoltre, i computer quantistici devono garantire che i qubits siano isolandoti fisicamente dalle interferenze esterne, mantenendoli a temperature molto basse o sottoponendoli a impulsi di energia attentamente controllati. Ecco perché l'interno del computer quantistico di D-Wave Systems ci sono -460 gradi Fahrenheit. Infine, sono necessari qubit aggiuntivi per correggere errori che si insinuano nel sistema, di conseguenza la realizzazione di un computer quantistico ha molte complicazioni.

1.3.2 Riduzione della complessità con il computer quantistico

Con i qubit e le porte quantistiche (Gates) [9], si possono progettare nuovi algoritmi, diversi da quelli digitali e analogici classici. L'obbiettivo è trovare soluzioni a problemi che sono intrattabili per i computer classici, come quando si vuole determinare una proprietà globale di una funzione.
Ad esempio, si vuole trovare il valore di un parametro x per cui la funzione $f(x)$ ha valore minimo o, se la funzione è periodica, il periodo della funzione che ci dà $f(x)$ minimo. Un algoritmo su un computer digitale potrebbe utilizzare un processo in cui $f(x)$ viene calcolato per una varietà di input diversi al fine di ottenere informazioni sufficienti sulla proprietà globale. Con un computer quantistico, invece, la possibilità di creare stati di sovrapposizione permette di calcolare il valore della funzione per molti possibili input simultaneamente. Ciò consentirà di rivelare la proprietà globale di cui si ha bisogno in un tempo molto minore.

Questa descrizione generale illustra il funzionamento di molti di molti algoritmi quantistici che sono già stati scoperti. Un esempio evidente è l'algoritmo di Grover [10], che riduce la complessità della ricerca attraverso N elementi da $O(N)$ a O $(N^(1/2))$. Questa accelerazione quadratica potrebbe essere utile in molte applicazioni come problemi di ottimizzazione e apprendimento automatico [11].

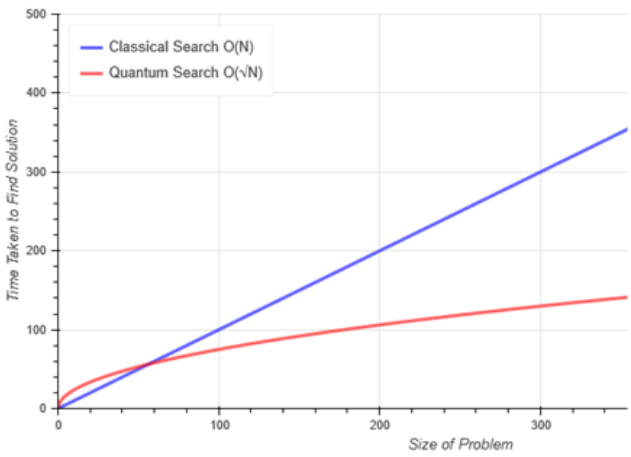


Figura 1.4. Confronto tra la complessità di una ricerca quantistic a e una classica. Immagine tratta da qiskit.org

Si può concludere dicendo che per un problema abbastanza grande, l'algoritmo di ricerca quantistica supererà sempre l'algoritmo di ricerca classico. Una velocità ancora più impressionante si ottiene con l'algoritmo di Shor, che analizza il

problema della fattorizzazione [6]. Ciò consente una soluzione quantistica per la fattorizzazione di numeri a n cifre con complessità $O(n^3)$. Si tratta di un aumento della velocità super polinomiale rispetto alla complessità dei computer digitali, che invece è di O $(e^{(n^{(1/3)})})$. Gli algoritmi quantistici verranno trattati nei capitoli successivi.

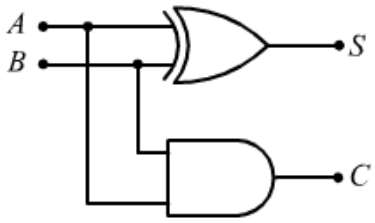
1.4 Dal computer classico al quantistico

I computer quantistici si basano sulla stessa idea di base dei computer digitali standard. La differenza principale è che usano i qubit, un'estensione del bit alla meccanica quantistica.

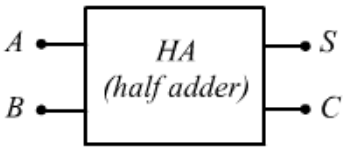
Sia che si utilizzino qubit o bit, bisogna manipolarli per trasformare i dati in input negli output di cui si ha bisogno. E' utile rappresentare questo processo in uno schema noto come schema circuitale. Vi sono ingressi a sinistra, uscite a destra e operazioni rappresentate da simboli nel mezzo chiamate "porte". Si vuole ad esempio eseguire la somma di due bit.

$$0 + 0 = 00$$
$$0 + 1 = 01$$
$$1 + 0 = 10$$
$$1 + 1 = 11$$

Per poter eseguire la sommatoria c'è bisogno un circuito con due porte, XOR e AND.



A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



$S=A \oplus B$
$C=A \cdot B$

Figura 1.5. Porte logiche AND e OR. Immagine tratta da www.edutecnica.it

S è dato dallo XOR di A e B, e C dall'AND di A e B.

Per i computer quantistici viene usata la stessa idea di base ma vi sono convenzioni diverse su come rappresentare input, output e simboli usati per le operazioni. Verranno utilizzati gli strumenti offerti dalla libreria Qiskit, che risulta essere la libreria scelta per la risoluzione del problema proposto nel terzo capitolo.
In questo capitolo, quindi, verranno mostrati i gate e i circuiti implementati in Qiskit per spiegare la manipolazione dei qubits e descrivere la differenza con i classici bit.

Di seguito il circuito quantistico che rappresenta lo stesso processo di cui sopra.

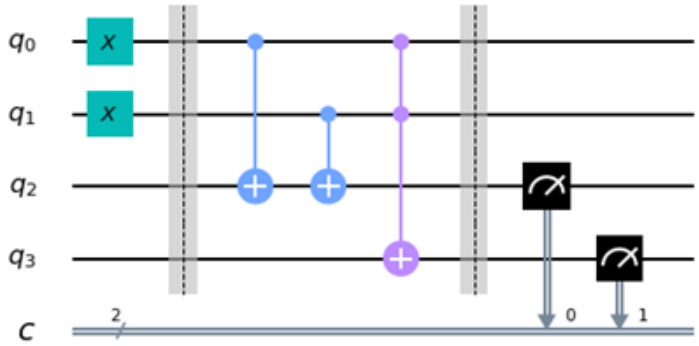


Figura 1.6. Circuito quantistico che esegue la somma

Inizialmente i qubit di input sono sempre settati a 0. Per poter cambiare l'input usiamo il NOT gate. Questo gate capovolge il valore del bit, quindi da 0 passa a 1 e viceversa ed è rappresentato dalla X in figura. Il simbolo nero posto alla fine invece serve per estrarre il risultato. La parte centrale è costituita dai gate che eseguono le operazioni richieste.

I due bit in input sono codificati nei qubit q_0 e q_1. L'esempio sopra applica il NOT gate ad entrambi, quindi gli imposta il valore 1 di conseguenza cerca di trovare la soluzione di 1 + 1. Il risultato sarà un numero di due bit, i cui valori verranno letti dai qubit q_2 e q_3.

Nei risultati della sommatoria il bit più a destra, quindi il valore salvato in S, è dato dallo XOR di A e B. Applicando la porta XOR, se i due bit sono uguali, il risultato sarà 0, se invece sono diversi il risultato sarà uno 1.

Input1	Input2	XOR Output
0	0	0
0	1	1
1	0	1
1	1	0

Nei computer quantistici, il lavoro del gate XOR è svolto dal controlled-NOT gate. Poiché è un nome piuttosto lungo, di solito viene chiamato semplicemente CNOT. In Qiskit il suo nome è CX. Negli schemi circuitali, è disegnato come nell'immagine sottostante.

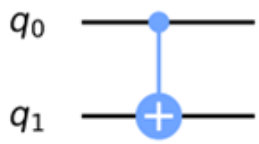


Figura 1.7. Gate CNOT

Viene applicato a una coppia di qubit. Uno funge da qubit di controllo (il primo dall'alto), l'altro funge da qubit di destinazione (il secondo). Il CNOT guarda i suoi due bit di input per verificare se sono uguali o diversi. Successivamente, sovrascrive il qubit di destinazione con uno 0 se sono uguali o con un 1 se sono diversi. Un esempio di circuito che testa il CNOT con l'ingresso 01 è il seguente.

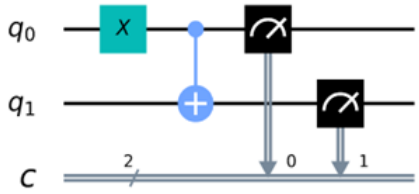


Figura 1.8. Circuito quantistico con CNOT

In $C[0]$ sarà settato in questo caso a 1 in quanto è stato applicato un NOT gate a q_0, che ha sempre valore iniziale 0, mentre $C[1]$ sarà settato a 1 in quanto

viene eseguito lo XOR tra 0 e 1. Essendo diversi ci sarà un 1, quindi il risultato di questo circuito è 11. Per l'operazione di somma che si vuole ottenere non bisogna sovrascrivere uno dei due qubit di input. Quindi, si scriverà il risultato su una diversa coppia di qubit. Vengono utilizzati quindi due CNOT (o CX).

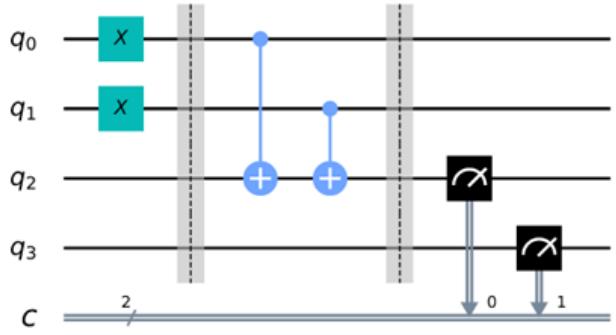


Figura 1.9. Circuito che esegue la somma

Con un circuito come quello in figura, ponendo in input le 4 combinazioni possibili di zeri e uni per i q_0 e q_1, si avrà:

Input0	Input1	Output2	Output3
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	0

In Qiskit i valori in output si leggono da destra a sinistra. I risultati saranno quindi:

Input0	Input1	Output2
0	0	00
0	1	01
1	0	01
1	1	00

Si noti che, il quarto caso non risulta essere la somma dei valori in input, poiché si dovrebbe avere un 1 in q_3 invece che uno 0. Per risolvere questo problema, si potrebbe semplicemente controllare se entrambi gli input sono settati a 1. Se lo sono, bisogna applicare un gate sull'ultimo qubit, il cui valore sarà settato a 1 solo se in input si hanno due uni. Si avrà in questo modo l'output desiderato.

Il nuovo gate di cui si ha bisogno è simile a un CNOT ma controllato su due qubit invece che uno solo. Eseguirà un NOT sul qubit target (q_3) solo quando entrambi gli input sono nello stato 1. Questo gate è chiamato Toffoli ed è fondamentalmente una porta AND.

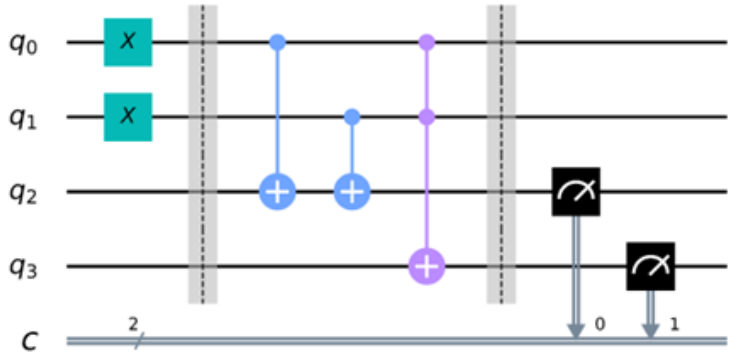


Figura 1.10. Gate Toffoli

Quest'ultimo circuito avrà in output la somma dei valori di input.

Input0	Input1	Output2
0	0	00
0	1	01
1	0	01
1	1	10

Con le porte NOT, CNOT e Toffoli (AND), si possono creare programmi che manipolano qualsiasi insieme di numeri di qualsiasi dimensione.

Queste tre porte sono sufficienti per poter eseguire qualsiasi tipo di operazione. La porta NOT permette di settare i bit in input con valore 1. La porta Toffoli è essenziale in quanto è l'elemento più semplice, da cui si può ricavare ogni altra tecnica di problem solving.

1.5 Lo stato del Qubit

1.5.1 La rappresentazione del qubit

I bit classici hanno sempre uno stato ben definito: 0 o 1.
Non ci sono altre dettagli che si possono aggiungere allo stato di un bit. Quindi,

per scrivere lo stato di un bit classico si possono semplicemente usare questi due valori binari.

Questa restrizione non è più presente per i bit quantistici.
Il qubit si trova in uno stato più complesso. Se non è ancora misurato può trovarsi in uno stato di sovrapposizione quantistica di 0 e 1. Una volta misurato può avere un valore che è 0 o 1.
Per capire come rappresentare gli stati quantistici, si analizzano i due casi più semplici.
È possibile preparare un qubit in uno stato per il quale dà sicuramente il risultato 0 quando misurato. Allo stesso modo, esiste uno stato del qubit che sicuramente produrrà un 1.

Questi due stati si escludono a vicenda. Quando viene misurato il qubit o emette sicuramente uno 0 o sicuramente un 1, non c'è sovrapposizione. Un modo per rappresentare questi due casi con la matematica è usare due vettori ortogonali.

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{1.1}$$

Precisamente, la notazione con | e ⟩ serve per esplicitare che si sta parlando di vettori che rappresentano gli stati dei qubit e quindi aiuta a distinguerli dai bit classici o da numeri 0 e 1.
Di conseguenza $|0\rangle$ rappresenta lo stato 0 del qubit , mentre $|1\rangle$ rappresenta lo stato 1.

Questi due vettori formano una base per lo spazio vettoriale che descrive lo stato del qubit. Ciò significa che qualsiasi stato quantum può essere scritto come la somma di questi vettori di base. Ad esempio:

$$|q_0\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{i}{\sqrt{2}}|1\rangle \tag{1.2}$$

in particolare

$$|q_0\rangle = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{i}{\sqrt{2}} \end{bmatrix} \tag{1.3}$$

Il vettore $|q0\rangle$ è chiamato vettore di stato del qubit.
È un vettore colonna di unità Norm, cioè la somma dei quadrati dei suoi elementi è 1, ed include tutte le informazioni necessarie per descrivere lo stato di un qubit, così come un singolo bit include tutte le informazioni necessarie per descrivere lo stato di una variabile binaria.
Per ora si è in grado di trarre solo alcune semplici conclusioni su questo particolare esempio di vettore di stato: non è interamente $|0\rangle$ e non è interamente $|1\rangle$. E' una combinazione lineare dei due.
Si trova quindi in uno stato di "sovrapposizione", che non risulta essere uno stato

ben definito.
In generale, quindi, il vettore di stato è così descritto:

$$|q\rangle = \alpha|0\rangle + \beta|1\rangle \tag{1.4}$$

la cui norma è

$$\sqrt{|\alpha|^2 + |\beta|^2} = 1 \tag{1.5}$$

1.5.2 Misurazione del qubit

Per trovare la probabilità di misurare uno stato $|\psi\rangle$ nello stato $|x\rangle$ si applica la seguente formula:

$$p(|x\rangle) = |\langle x|\psi\rangle|^2 \tag{1.6}$$

I simboli $\langle$ e $|$ ci dicono che x è un vettore riga.
Nella meccanica quantistica i vettori colonna ($|$,$\rangle$) vengono chiamate kets, mentre i vettori riga ($\langle$, $|$) bras. Insieme formano la notazione bra-ket. Qualsiasi ket ha un corrispondente bra. Nell'equazione precedente, $|x\rangle$ può rappresentare qualsiasi stato del qubit. Per trovare la probabilità di misurare $|x\rangle$, si prende il prodotto tra $|x\rangle$ e lo stato che si sta misurando (in questo caso $|\psi\rangle$), quindi si esegue il quadrato del valore assoluto.
Anche se può sembrare un po 'complicato, presto sarà più chiaro.
Considerando lo stato $|qo\rangle$ di prima, si calcola la probabilità di misurare $|0\rangle$, che risulta essere 0,5:

$$|q_0\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{i}{\sqrt{2}}|1\rangle \tag{1.7}$$

$$\langle 0|q_0\rangle = \frac{1}{\sqrt{2}}\langle 0|0\rangle - \frac{i}{\sqrt{2}}\langle 0|1\rangle = \frac{1}{\sqrt{2}} \times 1 - \frac{1}{\sqrt{2}} \times 0 = \frac{1}{\sqrt{2}} \tag{1.8}$$

$$|\langle 0|q_0\rangle|^2 = \frac{1}{2} \tag{1.9}$$

Allo stesso modo, la probabilità di misurare $|1\rangle$ risulta essere 0,5.
Questa regola governa il modo in cui si ottengono informazioni dagli stati quantistici. È quindi molto importante per tutto ciò che viene eseguito nel calcolo quantistico.
Quando viene misurato un qubit con vettore di stato $\begin{bmatrix}\alpha\\ \beta\end{bmatrix}$, si ottiene il risultato 0 con probabilità α^2 e il risultato 1 con probabilità β^2 . Sul risultato 0, il nuovo stato di qubit è pari a $\begin{bmatrix}1\\ 0\end{bmatrix}$; nel risultato 1 il suo stato è $\begin{bmatrix}1\\ 0\end{bmatrix}$.

1.5.3 Gli effetti della misurazione

I valori contenuti nel vettore di stato contengono informazioni sulla probabilità di trovare il qubit in uno stato specifico. Per sapere con certezza in quale stato si trova il qubit bisogna misurarlo. Ad esempio, se si misura un qubit nello stato:

$$|q\rangle = \alpha|0\rangle + |\beta|1\rangle \tag{1.10}$$

E il risultato risulta essere lo stato stato $|0\rangle$, nel momento in cui si misura di nuovo il qubit, c'è una probabilità del 100% di trovare di nuovo lo stato $|0\rangle$.

$$|q\rangle = \begin{bmatrix}\alpha\\ \beta\end{bmatrix} \longmapsto |q\rangle = |0\rangle = \begin{bmatrix}1\\ 0\end{bmatrix} \tag{1.11}$$

Una delle principali proprietà della misurazione è che non danneggia necessariamente i vettori di stato quantistici. Se si misura un qubit con stato $\begin{bmatrix}1\\0\end{bmatrix}$, che corrisponde allo stato 0, la misurazione di questo stato produrrà sempre il risultato 0.
In questo senso, se sono presenti solo bit classici (ad esempio, qubits $\begin{bmatrix}1\\0\end{bmatrix}$ o $\begin{bmatrix}0\\1\end{bmatrix}$), la misurazione non danneggia il sistema. Ciò significa che è possibile replicare i dati classici e manipolarli in un computer quantistico come in un computer classico [12].
La situazione cambia nel momento in cui lo stato iniziale del qubit non è uno stato ben definito (0 o 1), ma uno stato di sovrapposizione.

$$|q_0\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{i}{\sqrt{2}}|1\rangle \tag{1.12}$$

Q_0 si trova in uno stato che ha probabilità 1/2 di essere 1 e 1/2 di essere 0 quando viene eseguita la misurazione. Quindi, non si può sapere con sicurezza quale sarà il suo stato finale una volta misurato.
Tuttavia, sicuramente sarà 1 o 0 e non uno stato intermedio.
Se misurassimo costantemente ciascun qubit per tenere traccia del loro valore in ogni punto di un calcolo, sarebbero sempre in uno stato definito di $|0\rangle$, o $|1\rangle$. In quanto tali, non sarebbero diversi dai bit classici e il calcolo potrebbe essere facilmente sostituito da un calcolo classico. Per ottenere un calcolo quantistico bisogna consentire ai qubit di esplorare stati più complessi. Di conseguenza le misurazioni vengono quindi utilizzate solo quando è necessario estrarre un output. Ciò significa che spesso tutte le misurazioni vengono posizionate alla fine del circuito quantistico.

1.5.4 Rappresentazione geometrica

È possibile visualizzare un qubit in maniera chiara attraverso una rappresetazione geometrica.
Utilizzando una sfera di raggio unitario, gli stati del qubit verranno collocati in punti precisi della superficie della sfera, associando quindi ad ogni stato un punto. Lo stato 1 verrà collocato nel polo sud, lo stato 0 nel polo nord. Le altre locazioni indicano gli stati di sovrapposizioni quantistiche di 0 e 1. (Figura 1.11)

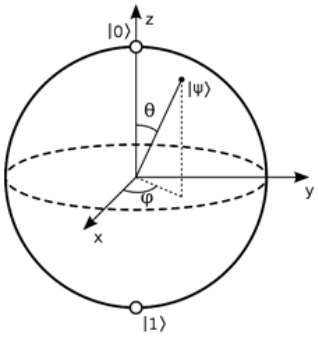


Figura 1.11. Sfera di Bloch. Immagine tratta da Wikipedia

Questa sfera è nota come la sfera di Bloch.
Molte delle operazioni su un singolo qubit che verranno approfondite nei capitoli successici possono essere visualizzate all'interno di questa sfera.
Per capire meglio la differenza la differenza tra un bit classico e un qubit, si osservi la figura 1.12

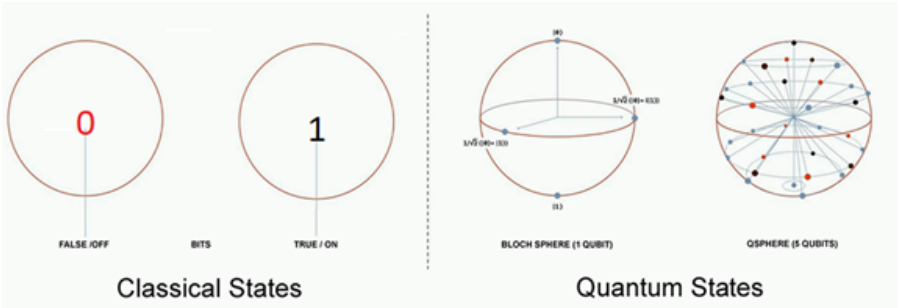


Figura 1.12. Stati classici e quantistici. Immagine tratta da www.pgs.com

Come si osserva, gli stati classici sono 0 o 1, lo stato di un qubit invece può essere una sovrapposizione di 1 e 0.
Esiste una corrispondenza biunivoca tra un generico stato di un qubit

$$|q\rangle = \alpha|0\rangle + \beta|1\rangle \tag{1.13}$$

e un punto p sulla sfera unitaria in R^3

$$p = \cos\theta/2|0\rangle + e^{(i\phi)}\sin\theta/2|1\rangle \tag{1.14}$$

Dove ϕ e θ sono le coordinate sferiche del punto.
Si può quindi scrivere:

$$|q\rangle = \alpha|0\rangle + e^{i\phi}\beta|1\rangle \tag{1.15}$$

Dato che il vettore di stato ha norma 1

$$\sqrt{|\alpha|^2 + |\beta|^2} = 1 \tag{1.16}$$

Si usa l'identità trigonometrica:

$$\sqrt{\sin^2 x + \cos^2 x} = 1 \tag{1.17}$$

per descrivere α e β reali in termini della variabile θ:

$$\alpha = \cos\frac{\theta}{2}, \beta = \sin\frac{\theta}{2} \tag{1.18}$$

Da questo si può descrivere lo stato di ogni qubit usando le due variabili ϕ e θ:

$$|q\rangle = \cos\frac{\theta}{2}|0\rangle + e^{i\phi}\sin\frac{\theta}{2}|1\rangle \tag{1.19}$$

1.5.5 Visualizzazione del qubit sulla sfera di Bloch

Interpretando θ e ϕ come coordinate sferiche (r = 1, poiché l'ampiezza dello stato del qubit è 1), si può tracciare qualsiasi stato del qubit sulla superficie della sfera di Bloch.

In figura 1.13 Vengono visualizzati i seguenti vettori di stato del qubit:

- $\begin{bmatrix}1\\0\end{bmatrix}$ cioè lo stato $|0\rangle$ con $\theta = 0$ e $\varphi = 0$
- $\begin{bmatrix}0\\1\end{bmatrix}$ cioè lo stato $|1\rangle$ con $\theta = 180$ e $\varphi = 0$

I successivi vettori rappresentano uno stato di sovrapposizione

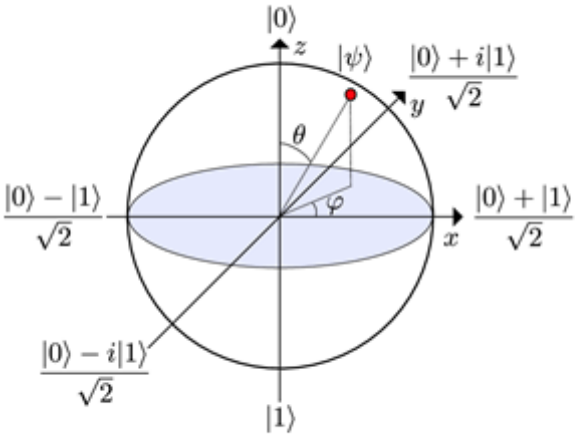


Figura 1.13. Visualizzazione dei qubit. Immagine tratta da *Quantum optics with artificial atoms*, Anton Frisk Kockum, 2014

- $\begin{bmatrix} 1/\sqrt{2} \\ i/\sqrt{2} \end{bmatrix}$ con $\theta = \pi/2$ e $\varphi = \pi/2$
- $\begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$ con $\theta = \pi/2$ e $\varphi = 0$, chiamato anche stato $|+\rangle$
- $\begin{bmatrix} 1/\sqrt{2} \\ -i/\sqrt{2} \end{bmatrix}$ con $\theta = \pi/2$ e $\varphi = 3\pi/2$
- $\begin{bmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix}$ con $\theta = 3\pi/2$ e $\varphi = 0$, chiamato anche stato $|-\rangle$

Dato che in input i qubit hanno sempre stato $|0\rangle$, per poter operare sui qubit e ottenere degli stati differenti bisogna ruotare gli assi cardinali con appositi Gates.

1.6 I gates

Nella sezione precedente sono stati esaminati tutti i possibili stati in cui potrebbe trovarsi un qubit. In particolare, ogni stato è descritto dalla seguente espressione:

$$|q\rangle = \cos\frac{\theta}{2}|0\rangle + e^{i\phi}\sin\frac{\theta}{2}|1\rangle \tag{1.20}$$

Dove θ e φ sono numeri reali.
In questa sezione verranno trattati i gate, cioè quelle operazioni che cambiano lo stato di un qubit [9].

1.6.1 X-GATE

L'X-gate è rappresentato dalla matrice Pauli-X:

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = |0\rangle\langle 1| + |1\rangle\langle 0| \tag{1.21}$$

Per vedere l'effetto che un gate ha su un qubit, si esegue semplicemente la moltiplicazione tra il vettore di stato del qubit e il gate. Si può vedere che l'X-gate cambia i valori del vettore di stato.

$$X|0\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = |1\rangle \tag{1.22}$$

Applicando in input quindi un qubit con stato 0, in output ci sarà un qubit con stato 1. Per capire come cambia la raffigurazione del qubit sulla sfera di Bloch, si osservi l'immagine seguente

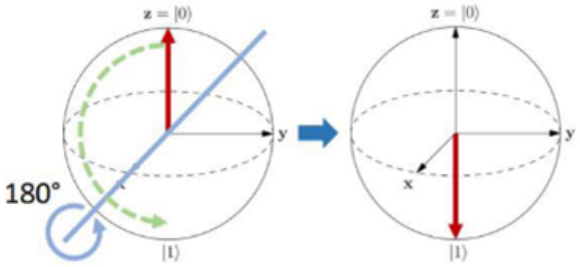


Figura 1.14. Visualizzazione dell'applicazione del Gate X

In pratica viene ruotato il qubit di 180° attorno all'asse x.
In Qiskit il Gate è rappresentato da una X

Figura 1.15. Gate X in Qiskit

In base all'input, gli output sono i seguenti:

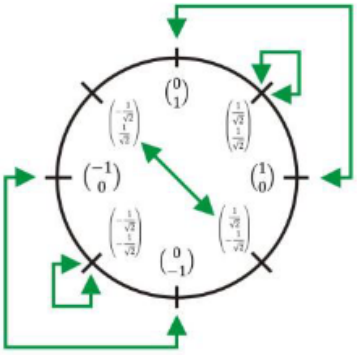


Figura 1.16. Visualizzazione dell'applicazione del Gate X

- $\begin{bmatrix}0\\1\end{bmatrix} \leftrightarrow \begin{bmatrix}1\\0\end{bmatrix}$
- $\begin{bmatrix}-1\\0\end{bmatrix} \leftrightarrow \begin{bmatrix}0\\-1\end{bmatrix}$
- $\begin{bmatrix}1/\sqrt{2}\\1/\sqrt{2}\end{bmatrix} \leftrightarrow \begin{bmatrix}1/\sqrt{2}\\1/\sqrt{2}\end{bmatrix}$
- $\begin{bmatrix}-1/\sqrt{2}\\-1/\sqrt{2}\end{bmatrix} \leftrightarrow \begin{bmatrix}-1/\sqrt{2}\\-1/\sqrt{2}\end{bmatrix}$
- $\begin{bmatrix}-1/\sqrt{2}\\1/\sqrt{2}\end{bmatrix} \leftrightarrow \begin{bmatrix}1/\sqrt{2}\\-1/\sqrt{2}\end{bmatrix}$

1.6.2 Y-GATE

L'Y-gate è rappresentato dalla matrice Pauli-Y:

$$Y = \begin{bmatrix}0 & i\\ i & 0\end{bmatrix}, Y = -i|0\rangle\langle 1| + i|1\rangle\langle 0| \tag{1.23}$$

In Qiskit è rappresentato da una Y

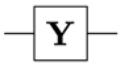


Figura 1.17. Gate Y in Qiskit

L'effetto è quello di ruotare il qubit di 180° attorno all'asse y

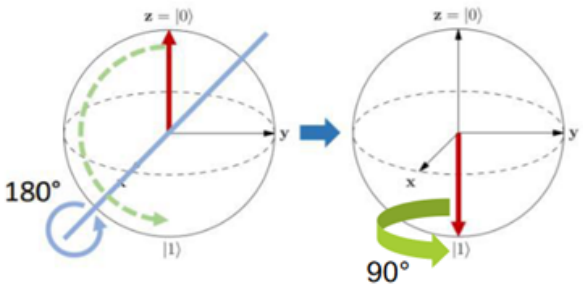


Figura 1.18. Visualizzazione dell'applicazione del Gate Y

1.6.3 Z-GATE

Lo Z-gate è rappresentato dalla matrice Pauli-Z:

$$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, Z = |0\rangle\langle 0| - |1\rangle\langle 1| \tag{1.24}$$

In Qiskit è rappresentato da una Z

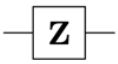


Figura 1.19. Gate Z in Qiskit

L'effetto è quello di ruotare il qubit di 180° attorno all'asse y

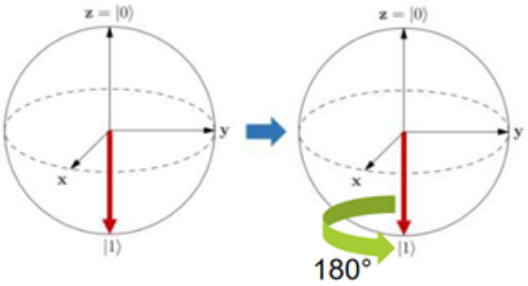


Figura 1.20. Visualizzazione dell'applicazione del Gate Z

1.6.4 HADAMARD GATE

La porta Hadamard (porta H) è una porta quantica fondamentale. Permette di allontanarci dai poli della sfera di Bloch e creare una sovrapposizione di $|0\rangle$ e $|1\rangle$. La matrice è la seguente:

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{1.25}$$

Ricordando che gli stati sull'asse X sono

$$|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \tag{1.26}$$

$$|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ -1 \end{bmatrix} \tag{1.27}$$

Applicando l'H gate agli stati 0 e 1 vengono eseguite le seguenti trasformazioni

$$H|0\rangle = |+\rangle \tag{1.28}$$

$$H|1\rangle = |-\rangle \tag{1.29}$$

Questo Gate quindi viene usato per passare a stati di sovrapposizione

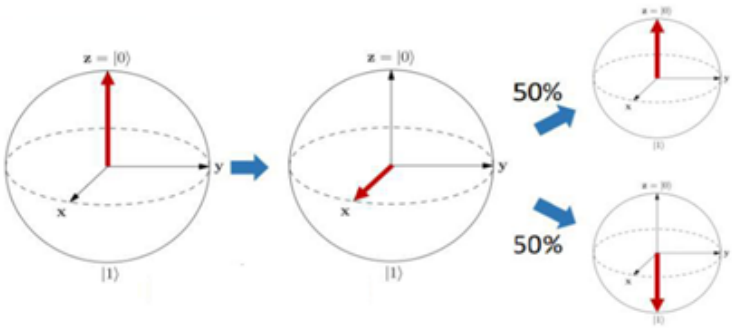


Figura 1.21. Visualizzazione dell'applicazione del Gate H

Lo stato $|+\rangle$ infatti, come lo stato $|-\rangle$, sono degli stati che attraverso la misurazione danno in output lo stato $|0\rangle$ o lo stato $|1\rangle$ con una probabilità del 50%. In Qiskit il Gate è rappresentato da una H (Figura 1.22)

In base all'input, gli output sono i seguenti

- $\begin{bmatrix} 0 \\ 1 \end{bmatrix} \leftrightarrow \begin{bmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix}$

Figura 1.22. Gate H in Qiskit

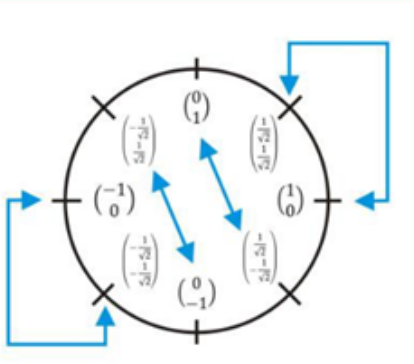


Figura 1.23. Input e Output applicando il gate H

- $\begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix} \leftrightarrow \begin{bmatrix} 1 \\ 0 \end{bmatrix}$
- $\begin{bmatrix} -1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix} \leftrightarrow \begin{bmatrix} 0 \\ -1 \end{bmatrix}$
- $\begin{bmatrix} -1 \\ 0 \end{bmatrix} \leftrightarrow \begin{bmatrix} -1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix}$

1.6.5 Rϕ-GATE

La porta $R\phi$ è parametrizzata, cioè ha bisogno di un numero (ϕ) per dirgli esattamente cosa fare. Il gate $R\phi$ esegue una rotazione di ϕ attorno all'asse Z (a volte è anche noto come gate Rz).
La sua matrice è la seguente:

$$R_\phi = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\phi} \end{bmatrix} \tag{1.30}$$

In base al valore assunto da π, vengono a crearsi altre tre porte.
Di seguito la porta I, S e T.

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, S = \begin{bmatrix} 1 & 0 \\ 0 & e^{\frac{i\pi}{2}} \end{bmatrix}, T = \begin{bmatrix} 1 & 0 \\ 0 & e^{\frac{i\pi}{4}} \end{bmatrix} \tag{1.31}$$

Vi è inoltre un altro Gate importante e fondamentale per capire l'entanglement. È il CNOT ed è applicato a una coppia di qubit. Viene approfondito nel paragrafo successivo.

1.7 L'Entanglement

Agli albori della meccanica quantistica, fu evidente ai matematici come Albert Einstein che le particelle quantistiche formavano correlazioni tra loro.
Lo stato quantistico di una particella (come la direzione del suo spin) poteva essere determinato attraverso lo stato di un'altra particella, indipendentemente dalla distanza tra le due.

Tale fenomeno fu definito da Einstein "azione spettrale a distanza" [13] e fece supporre che doveva esserci qualcosa di sbagliato nella meccanica quantistica.
Gli esperimenti successivi hanno dimostrato, invece, che tali correlazioni esistono in natura e sono al centro della meccanica quantistica.

Il concetto di entanglement si basa sull'assunzione che gli stati quantici di due particelle microscopiche, A e B inizialmente interagenti, risultano legati tra loro (Entangled in inglese significa legati o intrecciati) .
Anche se le due particelle si trovano a grande distanza l'una dall'altra, una possibile modifica sullo stato della particella A istantaneamente avrebbe un effetto misurabile sullo stato della particella B, determinando in tal modo il fenomeno della cosiddetta "azione fantasma a distanza" (spooky action at distance).

Nella fisica quantistica, quindi, le particelle entangled rimangono collegate in modo che le azioni eseguite su una particella influenzano l'altra.
L'entanglement si verifica quando una coppia di particelle, come i fotoni, interagisce fisicamente.

Quando un raggio laser viene sparato attraverso il cristallo BBO, i singoli fotoni possono essere divisi in coppie di fotoni intrecciati.
Le regole della fisica quantistica affermano che un fotone non osservato esiste contemporaneamente in tutti gli stati possibili, ma quando osservato o misurato mostra solo uno stato.
I fotoni possono essere separati e posizionati a una grande distanza l'uno dall'altro, centinaia di miglia o anche di più, tuttavia quando il fotone A viene osservato ed assume uno stato di rotazione verso l'alto, il fotone B, "intrecciato", assume uno stato relativo a quello del fotone A (in questo caso, uno stato di rotazione verso il basso). Il trasferimento di stato tra il fotone A e il fotone B avviene a una velocità

di almeno 10.000 volte la velocità della luce, indipendentemente dalla distanza.
La massima distanza testata sperimentalmente è stata di circa 310 miglia (500 chilometri), corrispondente alla distanza dalla terra della Stazione Spaziale Internazionale in orbita.

Per comprendere il fenomeno a livello matematico, si procede per gradi.

Si è visto che un singolo bit ha due stati possibili e lo stato di un qubit ha due ampiezze complesse.
Allo stesso modo, due bit hanno quattro stati possibili: 00-01-10-11. Per descrivere lo stato di due qubit sono necessarie quattro ampiezze complesse.
Si memorizzano le ampiezze in un vettore 4D in questo modo:

$$|a\rangle = a_{00}|00\rangle + a_{01}|01\rangle + a_{10}|10\rangle + a_{11}|11\rangle = \begin{bmatrix} a_{00} \\ a_{01} \\ a_{10} \\ a_{11} \end{bmatrix} \tag{1.32}$$

le regole di misurazione sono le medesime:

$$p(|00\rangle) = |\langle 00|a\rangle|^2 = |a_{00}|^2 \tag{1.33}$$

La probabilità di misurare lo stato $|00\rangle$ è dato dal modulo del valore di a00 al quadrato. Valgono le stesse implicazioni, come la condizione di normalizzazione:

$$|a_{00}|^2 + |a_{01}|^2 + |a_{10}|^2 + |a_{11}|^2 = 1 \tag{1.34}$$

Con due qubit separati si possono descrivere il loro stato collettivo usando il prodotto tensoriale:

$$|a\rangle = \begin{bmatrix} a_0 \\ a_1 \end{bmatrix}, |b\rangle = \begin{bmatrix} b_0 \\ b_1 \end{bmatrix} \tag{1.35}$$

$$|ba\rangle = |b\rangle \otimes |a\rangle = \begin{bmatrix} b_0 \times \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} \\ b_1 \times \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} b_0 a_0 \\ b_0 a_1 \\ b_1 a_0 \\ b_1 a_1 \end{bmatrix} \tag{1.36}$$

Se ci fossero n qubit, si dovrebbe tener traccia di 2^n ampiezze complesse. Come si può vedere, questi vettori crescono esponenzialmente con il numero di qubit. Questo è il motivo per cui i computer quantistici con un gran numero di qubit sono così difficili da simulare.

E' stato visto precedentemente che un X-gate è rappresentato dalla matrice:

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = |0\rangle\langle 1| + |1\rangle\langle 0| \tag{1.37}$$

e che agisce sullo stato $|0\rangle$:

$$X|0\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = |1\rangle \tag{1.38}$$

Così come è stato utilizzato il prodotto tensoriale per calcolare i vettori di stato multi-qubit, si usa il prodotto tensore per calcolare le matrici che agiscono su questi vettori di stato.
Considerando il circuito seguente in figura 1.24

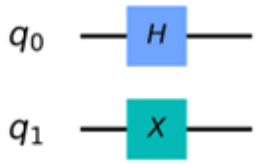


Figura 1.24. Applicazione del gate H e X

si possono rappresentare le operazioni simultanee (H e X) usando il loro prodotto tensoriale :

$$X|q_1\rangle \otimes H|q_0\rangle = (X \otimes H)|q_1 q_0\rangle \tag{1.39}$$

L'operazione si presenta così:

$$X \otimes H = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \otimes \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 0 \times \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} & 1 \times \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \\ 1 \times \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} & 0 \times \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \\ 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \end{bmatrix} \tag{1.40}$$

Si può quindi applicare al vettore di stato 4D $|qoq1\rangle$.
Il prodotto vettoriale si può anche scrivere così:

$$X \otimes H = \begin{bmatrix} 0 & H \\ H & 0 \end{bmatrix} \tag{1.41}$$

Per imparare come i qubit interagiscono tra loro, bisogna considerare un importante gate a due qubit, il gate CNOT. Questo gate è un gate condizionale che esegue un X-gate sul secondo qubit (target), se lo stato del primo qubit (control) è $|1\rangle$.
Il gate è disegnato in un circuito con q0 come controllo e q1 come target.

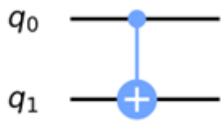


Figura 1.25. Gate CNOT

Quando i qubit non sono in stato di sovrapposizione (si comportano come bit classici) questa porta è molto semplice e intuitiva da capire. Si fa riferimento alla classica tabella di verità:

Input	Output
00	00
01	11
10	10
11	01

In questo caso, la matrice CNOT ha le seguenti matrici, a seconda di quale qubit è il controllo e quale è l'obiettivo:

$$CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}, CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{1.42}$$

La matrice sinistra corrisponde al CNOT nel circuito sopra. Questa matrice scambia le ampiezze di $|01\rangle$ e $|11\rangle$ nel vettore di stato:

$$|a\rangle = \begin{bmatrix} a_{00} \\ a_{01} \\ a_{10} \\ a_{11} \end{bmatrix}, CNOT|a\rangle = \begin{bmatrix} a_{00} \\ a_{11} \\ a_{10} \\ a_{01} \end{bmatrix} \tag{1.43}$$

Si veda ora come agisce su un qubit in stato di sovrapposizione. Si imposta un qubit nello stato $|+\rangle$ come mostrato nella figura 1.26

Di cui il vettore di stato risulta essere:

$$Statevector = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \\ 0 \\ 0 \end{bmatrix} \tag{1.44}$$

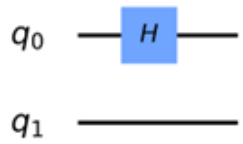


Figura 1.26. Applicazione gate H al qubit q0

Questo produce lo stato $|0\rangle \otimes |+\rangle = |0+\rangle$

$$|0+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |01\rangle) \tag{1.45}$$

Quando si applica il gate CNOT il circuito diventa:

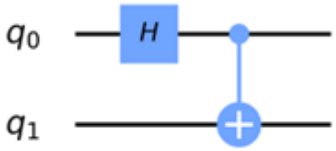


Figura 1.27. Circuito con gate H e CNOT

Il vettore di stato è uguale a quello precedente, invertendo il secondo valore con il quarto. Lo stato risultante è così descritto:

$$CNOT|0+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) \tag{1.46}$$

In questo stato i due qubit sono "entangled".
Questo stato è noto come uno stato di Bell, ed ha il 50% di probabilità di essere misurato nello stato $|00\rangle$ e il 50% di probabilità di essere misurato nello stato $|11\rangle$. La cosa più interessante è che ha una probabilità dello 0% di essere misurata negli stati $|01\rangle$ o $|10\rangle$.
Questo stato ha implicazioni interessanti.
Sebbene i qubit considerati siano in sovrapposizione, misurarne uno ci dirà lo stato dell'altro e ne farà collassare la sovrapposizione.
Ad esempio, se si misura un qubit e otteniamo lo stato $|1\rangle$, anche l'altro qubit quando verrà misurato ci darà lo stato $|1\rangle$.

Per capire cosa significa che i due qubit sono "intrecciati", si noti che misurare lo stato del primo qubit significa sapere il valore che assumerà l'altro qubit, in quanto

le combinazioni possibili sono solo 00 e 11.
Se si separano questi qubit ad anni luce di distanza, misurare un qubit fa crollare la sovrapposizione e sembra avere un effetto immediato sull'altro. Questa è l' "azione spettrale a distanza" che ha sconvolto così tanti fisici all'inizio del XX secolo.

È importante notare che il risultato della misurazione è casuale e le statistiche di misurazione di un qubit non sono influenzate da alcuna operazione sull'altro qubit. Per questo motivo, non è possibile utilizzare gli stati quantistici condivisi per comunicare. Questo è noto come teorema di non comunicazione [14].

Capitolo 2

Le applicazioni del computer quantistico

In questo capitolo si approfondiscono gli ambiti applicativi del computer quantistico.
La potenza del calcolo quantistico è impressionante, tuttavia ciò non significa che il quantum computing sia più efficiente per tutti i tipi di problemi. Di seguito sono elencate alcune delle principali applicazioni.

2.1 Le varie applicazioni

2.1.1 Intelligenza artificiale

Un'applicazione primaria è l'intelligenza artificiale.
Per la maggior parte delle applicazioni industriali dell'intelligenza artificiale, come il riconoscimento delle immagini o la previsione dei consumi, viene utilizzato l'apprendimento supervisionato.
Esso si basa sul principio dell'apprendere dall'esperienza, diventando più accurato con il feedback, fino a quando il programma sembra esibire "intelligenza".
Il feedback si basa sul calcolo delle probabilità per molte possibili scelte, e quindi per stati differenti.
Come detto in precedenza, una delle particolarità del nuovo sistema di calcolo è di consentire la rappresentazione di più stati quantistici contemporaneamente, il che è particolarmente conveniente quando si usano le tecniche di intelligenza artificiale. l'IA, quindi, è un candidato ideale per il calcolo quantistico. Promette di sconvolgere ogni settore, dagli autoveicoli alla medicina.
Usare la velocità quantistica per le attività di apprendimento automatico porterebbe a calcoli più veloci e algoritmi più efficienti in termini di risorse, inoltre consentirebbe all'IA di affrontare problemi che sono attualmente irrealizzabili a causa della loro complessità e dimensione.

Lockheed Martin, ad esempio, prevede di utilizzare il suo computer quantistico D-Wave per testare un software di pilota automatico che attualmente risulta troppo complesso per i computer classici [16].
Google, invece, sta utilizzando un computer quantistico per progettare software in grado di distinguere le automobili da altri oggetti.
Gli assistenti vocali potrebbero trarre grande vantaggio da questa implementazione, poiché quantum potrebbe contribuire in modo esponenziale a migliorare la loro precisione, aumentando sia la loro potenza di elaborazione che la quantità di dati che sarebbero in grado di gestire.
Il calcolo quantistico aumenta il numero di variabili di calcolo che le macchine possono destreggiare e quindi consente loro di fornire risposte più rapide, proprio come farebbe una persona.
Il team di ricerca quantistica di IBM ha recentemente scoperto che utilizzando qubit entangled per eseguire un esperimento di classificazione dei dati, il tasso di errore si è dimezzato rispetto a quello ottenuto usando qubit non districati.
La ricerca di IBM è arrivata sulla scia di un altro promettente algoritmo di classificazione dell'apprendimento automatico: un ibrido quantistico classico eseguito su una macchina a 19 qubit costruita da Rigetti Computing. È un passo importante e potrebbe portare ad assistenti virtuali "intelligenti" o personaggi dei videogiochi non controllati dal giocatore che si comportano in modo iperrealistico [18].
I potenziali progressi sono numerosi.
"Penso che l'IA possa accelerare il calcolo quantistico", ha detto Pichai di Google, "e il calcolo quantistico può accelerare l'IA" [19].

2.1.2 Modellistica Molecolare

L'universo funziona fondamentalmente in modo quantistico, quindi si potrà capire meglio la natura dell'universo grazie a questi nuovi computer.
Un altro tipo di applicazione, allora, è la modellazione delle interazioni molecolari; in particolare, la ricerca delle configurazioni ottimali per le reazioni chimiche.
La "chimica quantistica" è molto complessa e solo le molecole più semplici vengono analizzate dai computer digitali attuali.
I computer quantistici sviluppati finora, invece, non avrebbero alcuna difficoltà ad analizzare anche i processi chimici più complessi.
È una tecnologia nuova e ciò che rende la meccanica quantistica innovativa è la capacità di simulare molecole e processi molecolari. La scoperta di farmaci è un ottimo esempio [15].
Google ha già fatto incursione in questo campo simulando l'energia delle molecole di idrogeno. Le implicazioni vanno dalle celle solari alle droghe farmaceutiche, e in particolare alla produzione di fertilizzanti; poiché i fertilizzanti rappresentano il 2 percento del consumo globale di energia, le conseguenze per l'energia e l'ambiente sarebbero profonde.

Una società che si concentra sulla simulazione molecolare, in particolare sul comportamento delle proteine, è la startup biotecnologica di Toronto ProteinQure. Con un finanziamento iniziale di 4 milioni di dollari, collabora con i leader dell'informatica quantistica (IBM, Microsoft e Rigetti Computing) e con gruppi di ricerca farmaceutica (SRI International, AstraZeneca) per esplorare il potenziale di QC nella modellazione delle proteine.
Questa è una strada profondamente complicata ma ad alto rendimento per lo sviluppo di farmaci progettati per scopi medici mirati.
Come ha osservato il Boston Consulting Group [19], la semplice modellazione di una molecola di penicillina richiederebbe un computer classico incredibilmente grande, con bit di potenza da 10 a 86. Per i computer quantistici avanzati, tuttavia, lo stesso processo potrebbe essere un gioco da ragazzi e potrebbe portare alla scoperta di nuovi farmaci per malattie gravi come il cancro, l'Alzheimer e le malattie cardiache.

2.1.3 Crittografia

Molti aspetti importanti della sicurezza IT si basano sulla crittografia a chiave pubblica, i messaggi quindi vengono criptati.
Ciò è essenziale per il commercio elettronico e la protezione delle informazioni segrete.
Le tecniche di criptografia si basano su algoritmi matematici molto difficili da "rompere". Gli algoritmi moderni con lunghezze di chiave adeguate (ad es. AES-128, RSA-2048, ECDSA-256, ecc.) non sono suscettibili all'attacco di forza bruta e, anche con enorme potenza di calcolo, sarebbero necessari secoli o anche più tempo. Tuttavia, è possibile "romperli" con algoritmi univoci per computer quantistici (ad esempio "l'algoritmo di Shor").
Gli algoritmi simmetrici utilizzati per la crittografia, come AES, sono ancora considerati sicuri (con una lunghezza della chiave sufficiente - ad es. AES-256 o superiore); mentre, gli attuali algoritmi asimmetrici come RSA ed ECDSA saranno resi sostanzialmente inutili quando i computer quantistici raggiungeranno una certa scala.
Ciò interromperà quasi tutte le applicazioni pratiche della crittografia in uso oggi, rendendo l'e-commerce e molte altre applicazioni digitali, su cui facciamo affidamento nella nostra vita quotidiana, totalmente insicure.

Per affrontare questa minaccia, il National Institute of Standards and Technology (NIST) degli Stati Uniti, la cui carta è promuovere l'innovazione e la competitività industriale attraverso un ampio spettro di tecnologie e attività, inclusa la sicurezza informatica, ha avviato il processo di standardizzazione di nuovi algoritmi crittografici a chiave pubblica che non può essere attaccato in modo efficiente nemmeno con l'aiuto del computer quantistico. Con partecipanti da tutto il mondo, l'obiettivo di questo progetto è identificare nuovi algoritmi crittografici resistenti

agli attacchi dei computer quantistici e quindi standardizzarli per un ampio utilizzo [19].

2.1.4 Modello Finanziario

L'elenco dei partner che compongono il cosiddetto Quantum Network di Microsoft include una sfilza di università di ricerca e gruppi tecnici incentrati sul quantum, ma pochi preziosi affiliati aziendali.
I mercati moderni si basano su alcuni dei sistemi esistenti più complicati.
Mentre i sistemi scientifici e matematici sono in continuo sviluppo, il sistema finanziario soffre ancora di una grande quantità di problemi e a differenza degli altri campi scientifici non esiste un ambiente controllato in cui eseguire esperimenti.
Investitori e analisti, dunque, si sono rivolti al calcolo quantistico.
Il fatto che le società di servizi finanziari estremamente redditizie vogliano sfruttare la tecnologia che cambia paradigma non è certo uno shock.
Gli investitori spesso desiderano valutare la distribuzione dei risultati in un numero estremamente elevato di scenari generati casualmente. Questo lavoro viene svolto senza sforzi da un computer quantistico, inoltre i modelli quantistici e finanziari hanno una corrispondenza naturale grazie alle somiglianze strutturali, infatti un vantaggio immediato è che la casualità inerente ai computer quantistici è congruente alla natura stocastica dei mercati finanziari [16].
Un altro vantaggio offerto da quantum è che operazioni finanziarie come l'arbitraggio possono richiedere molti passaggi dipendenti dal percorso e il numero di possibilità supera rapidamente la capacità di un computer digitale, ritorna molto utile quindi l'utilizzo di un computer quantistico.
Affinché un settore finanziario trovi il giusto mix per investimenti fruttuosi basati sui rendimenti attesi, il rischio associato e altri fattori sono importanti per sopravvivere nel mercato. Per ottenere ciò, la tecnica delle simulazioni "Monte Carlo" viene continuamente eseguita su computer convenzionali, che però impiegano molto tempo prima di arrivare a una soluzione.
Applicando la tecnologia quantistica per eseguire questi calcoli massicci e complessi, le aziende possono non solo migliorare la qualità delle soluzioni, ma anche ridurre i tempi per svilupparle [15].
Di conseguenza molte ricerche si sono concentrate specificamente sul cercare di accelerare notevolmente il modello Monte Carlo, che essenzialmente misura la probabilità di vari risultati e i rischi corrispondenti.

Il trading algoritmico è un'altra potenziale applicazione in cui la macchina utilizza algoritmi complessi per attivare automaticamente le transazioni azionarie analizzando le variabili di mercato, il che è un vantaggio, soprattutto per le transazioni ad alto volume.

A parte la sua applicazione apparentemente chiara per la valutazione del rischio, il quantum nella finanza potrebbe avere un ampio futuro.

2.1.5 Previsioni del tempo

Attualmente, il processo di analisi delle condizioni meteorologiche con i computer tradizionali a volte può richiedere più tempo di quello necessario affinché avvenga la condizione climatica che si sta cercando di prevedere [15].
Le previsioni meteorologiche includono diverse variabili da considerare, come la pressione atmosferica, la temperatura e la densità dell'aria, il che rende difficile una previsione accurata.
La capacità di un computer quantistico di elaborare grandi quantità di dati in un breve periodo e l'applicazione dell'apprendimento automatico quantistico può aiutare a migliorare il riconoscimento dei modelli meteorologici, che a sua volta permetterà di prevedere i cambiamenti climatici in pochissimo tempo e con eccellente precisione [16]. In questo modo sarà possibile anche prevedere eventi meteorologici estremi e potenzialmente salvare migliaia di vite all'anno. Si avrà una visione più approfondita del cambiamento climatico con particolare attenzione al riscaldamento globale e si potranno adottare le misure adeguate ad attenuarne i danni.
Il Met Office del Regno Unito ha già investito molto nell'informatica quantistica per migliorare le previsioni, mentre IBM Research ha collaborato con The Weather Company, la University Corporation for Atmospheric Research (UCAR) e il National Center for Atmospheric Research (NCAR) in America per sviluppare un modello i scala delle tempeste, in rapido aggiornamento, in grado di prevedere i temporali a livello locale [17].
Questo modello è il primo a coprire l'intero globo e fornirà previsioni ad alta risoluzione anche nelle aree meno servite.

2.1.6 Fisica delle particelle

I modelli di fisica delle particelle sono spesso straordinariamente complessi e richiedono enormi quantità di tempo di calcolo per la simulazione numerica.
I ricercatori dell'Università di Innsbruck e dell'Istituto per l'ottica quantistica e l'informazione quantistica (IQOQI) hanno recentemente utilizzato un sistema quantistico programmabile per eseguire tale simulazione. Pubblicato su Nature, il team ha utilizzato una versione semplice del computer quantistico in cui gli ioni eseguivano operazioni logiche. Questa simulazione ha mostrato un eccellente accordo rispetto agli esperimenti reali della fisica.
"Questi due approcci si completano perfettamente a vicenda", afferma il fisico teorico Peter Zoller. "Non possiamo sostituire gli esperimenti effettuati con collisori di particelle. Tuttavia, sviluppando simulatori quantistici, un giorno potremmo essere in grado di comprendere meglio questi esperimenti " [16].

Sebbene il calcolo quantistico stia già influenzando i campi sopra elencati, l'elenco non è affatto esaustivo e come per tutte le nuove tecnologie, verranno sviluppate applicazioni attualmente inimmaginabili, man mano che l'hardware continua ad evolversi, e create nuove opportunità.

2.2 D-wave vs IBM

Proprio come una GPU elabora le istruzioni relative alla grafica in modo esponenzialmente più veloce di una CPU, alcuni problemi possono essere risolti più velocemente con algoritmi quantistici, sfruttando gli effetti quantistici di sovrapposizione ed l'entanglement.

D-Wave, il più famoso quantum annealer, e il computer quantistico a gate non sono concorrenti. Sebbene si basino sugli stessi concetti, hanno campi di utilizzo ben diversi.

D-wave è un progetto per scopi speciali che mira a problemi di ottimizzazione in cui l'interconnettività tra i "nodi" nel grafo, che rappresenta il problema, è limitata. I chip D-wave hanno quindi molti qubit, ma ognuno si accoppia solo a pochi dei suoi vicini. La programmazione avviene regolando il fattore di accoppiamento tra i qubit, per riflettere il peso degli archi nel grafo.
Inoltre, D-Wave si è concentrato sullo sviluppo della cosiddetta tecnologia annealer tramite cui i qubit vengono raffreddati durante l'esecuzione di un algoritmo; ciò consente di modificarne passivamente il valore.

Al contrario, i computer quantistici IBM Q e Google si basano sull'interconnessione di una serie di porte "universali", proprio come un FPGA collega insieme molte porte binarie classiche in modo programmabile. Almeno una di queste porte universali deve essere una porta a due ingressi, principalmente una porta CNOT (paragonabile a una porta exor).
L'attuale computer quantistico più grande di IBM contiene 65 qubit, contro 72 con Google.

Le tecnologie di Google e D-wave sono fondamentalmente molto diverse e attualmente il focus principale di aziende come IBM, Microsoft, Google è sui sistemi basati su gate.

Nel prossimo paragrafo si approfondisce il Quantum annealing nel computer D-wave per comprendere appieno le differenze tra le due tecnologie.

2.2.1 D-wave e quantum Annealing

La macchina D-Wave è un annealer quantistico che esegue algoritmi di calcolo quantistico adiabatico.
La ricottura quantica funziona meglio su problemi in cui ci sono molte potenziali soluzioni e nel trovare una soluzione "abbastanza buona" o "minima locale".
E' ottimo, quindi, per ottimizzare le soluzioni dei problemi cercando rapidamente il valore più basso.

Molti problemi interessanti possono essere ridotti a problemi di ottimizzazione nella ricerca del minimo. I fisici hanno in genere familiarità con questo tipo di problemi di minimizzazione dell'energia e hanno inventato modi intelligenti per trovare i minimi (punti più bassi). Molti di questi progressi sono dovuti a una tecnica chiamata Gradient Descent [20], in cui una funzione di costo viene ridotta al minimo.
La classe di problemi QUBO, o quadratica non vincolata, è una classe di problemi risolvibili dal Quantum Annealing.

La capacità di tradurre il problema in uno risolvibile su Quantum Hardware è cruciale e D-wave fornisce software adeguati a tale scopo.
Uno dei problemi che sono stati risolti su un Quantum Annealer come D-wave riguarda il flusso di traffico.
Nel 2019 durante una conferenza tecnologica di Lisbona, la Volkswagen ha collaborato per dirigere gli autobus in modo ottimizzato utilizzando le macchine D-wave. Il sistema ha calcolato il percorso più veloce per ciascuno dei nove autobus, individualmente e quasi in tempo reale [21].
I qubit sono collegati insieme in modo da creare una griglia di qubit. Ad esempio, il sistema D-wave One utilizzava 128 qubit. I sistemi più recenti utilizzano griglie molto più grandi, 2000 qubit per l'ultimo D-wave 2000.
Proprio come il numero di transistor sulle CPU è aumentato, così è aumentato il numero di qubit sui chip D-wave.
Le interazioni tra qubits svolgono un ruolo fondamentale per la formulazione di un problema.
Bisogna dunque codificare le interazioni tra qubits e dare un peso a ciascuno al fine di specificare il problema e vincoli.
Dal confronto con i tipici problemi di tipo QUBO, si può trovare una mappatura che può essere applicata alla rete di qubit. Successivamente il sistema può essere campionato e i campioni identificheranno le soluzioni con le energie più basse.
Inutile dire che questo sistema è probabilistico e dovranno verificarsi più esecuzioni per arrivare a una possibile soluzione.

2.2.2 IBM e i gates

I computer quantistici basati su gate potrebbero sembrare più familiari.
Nel dominio classico, è possibile eseguire alcune semplici operazioni su un registro a 8 bit.
Si prenda la seguente serie di bit 01100110 e si esegua un'operazione NOT gate bit per bit (traducendo 0 -> 1 e 0 -> 1), il risultato sarà: 10011001.
In effetti, si potrebbero eseguire tutti i tipi di operazioni tra bit, non solo NOT, il che rende molto versatile il computer classico.

Proprio come il computer classico, in un computer quantistico con gate è possibile applicare diversi gate ai qubit in numerose configurazioni.
Proprio come nel caso classico, anche nel mondo quantistico ci sono NOT gates che invertono lo stato di un qubit. Inoltre, ci sono molte altre porte che si possono applicare, come CNOT (a Controlled NOT), Hadamard, porte di rotazione (Rx) ecc (Riferimenti al paragrafo 1.6).
Tipicamente è possibile leggere l'andamento o l'evoluzione dei qubit da sinistra a destra partendo da una configurazione iniziale; le letture, o misurazioni, possono avvenire quando richiesto.

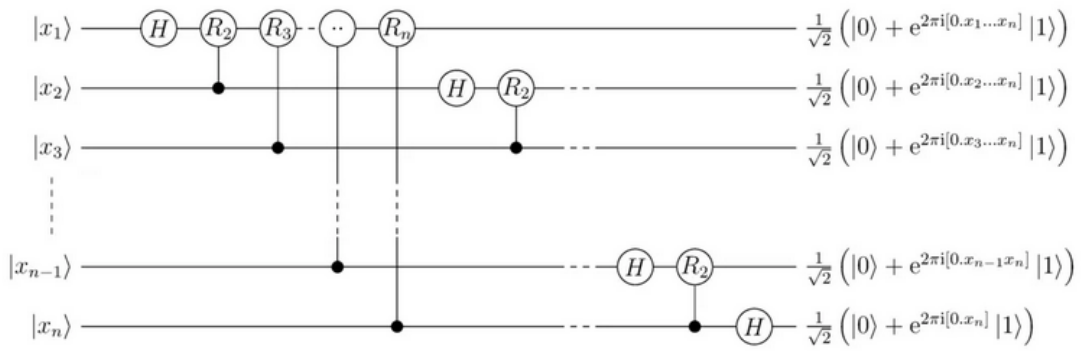


Figura 2.1. Esempio di circuito con gates. Immagine tratta da quantumzeitgeist.com

2.2.3 Differenza tra quantum annealing e gate

Ci sono differenze fondamentali tra quantum annealing e gate, anche se entrambe le architetture si basano sul qubit.
Il vantaggio dei Quantum annealer consiste in un numero maggiore di qubits rispetto ai sistemi basati su gate, a condizione che il problema venga manipolato per renderlo eseguibile sull'hardware.

Tuttavia, potrebbero esserci alcuni problemi di connessione nell'affrontare tanti qubit (2000) nell'ultima generazione di onde D e ciò potrebbe significare in futuro

un'esplosione di complessità.
Generalmente gran parte del mondo si concentra su circuiti quantistici basati su gate, tuttavia il Quantum annealer è più adatto con un problema di ottimizzazione adeguato.

2.3 La scelta del computer quantistico e della libreria

2.3.1 Perchè IBM

Per il progetto di tesi si è scelto il computer quantistico IBM in quanto il problema e la successiva ricerca della soluzione sono stati proposti e seguiti dall'azienda Blue-Reply.
Reply è partner IBM, quindi gli sviluppi quantum IBM sono di competenza dell'azienda. Inoltre, si ha accesso a materiale specifico riservato e prioritariamente a servizi che IBM espone.

2.3.2 Libreria utilizzata

La libreria utilizzata è Qiskit-aqua.
Aqua è situata in cima all'ecosistema Qiskit ed è una libreria open-source completamente scritta in Python e progettata per essere modulare ed estendibile a più livelli.
Come mostrato nella figura, questa flessibilità consente agli utenti con diversi livelli di esperienza e interessi scientifici di contribuire ad estendere Aqua in tutto lo stack.

Attualmente Aqua supporta quattro applicazioni, in domini che sono stati a lungo identificati come potenziali aree per il calcolo quantistico: Chimica, Intelligenza Artificiale (AI), Ottimizzazione e Finanza.
Nuovi domini possono essere facilmente aggiunti utilizzando le versatili interfacce Aqua.
A questo livello di applicazione, Aqua consente di sfruttare il software computazionale classico come front-end dell'applicazione quantistica, senza la necessità per l'utente finale di apprendere un nuovo linguaggio di programmazione.
Dietro le quinte, Aqua utilizza processi ibridi classico / quantistico; alcuni calcoli iniziali vengono eseguiti in modo classico e i risultati di tali calcoli vengono quindi combinati con dati di configurazione specifici del problema e tradotti in input per uno o più algoritmi quantistici.
Gli algoritmi Aqua funzionano su Terra, l'elemento di Qiskit responsabile della costruzione, compilazione ed esecuzione di circuiti quantistici su simulatori o dispositivi quantistici reali.

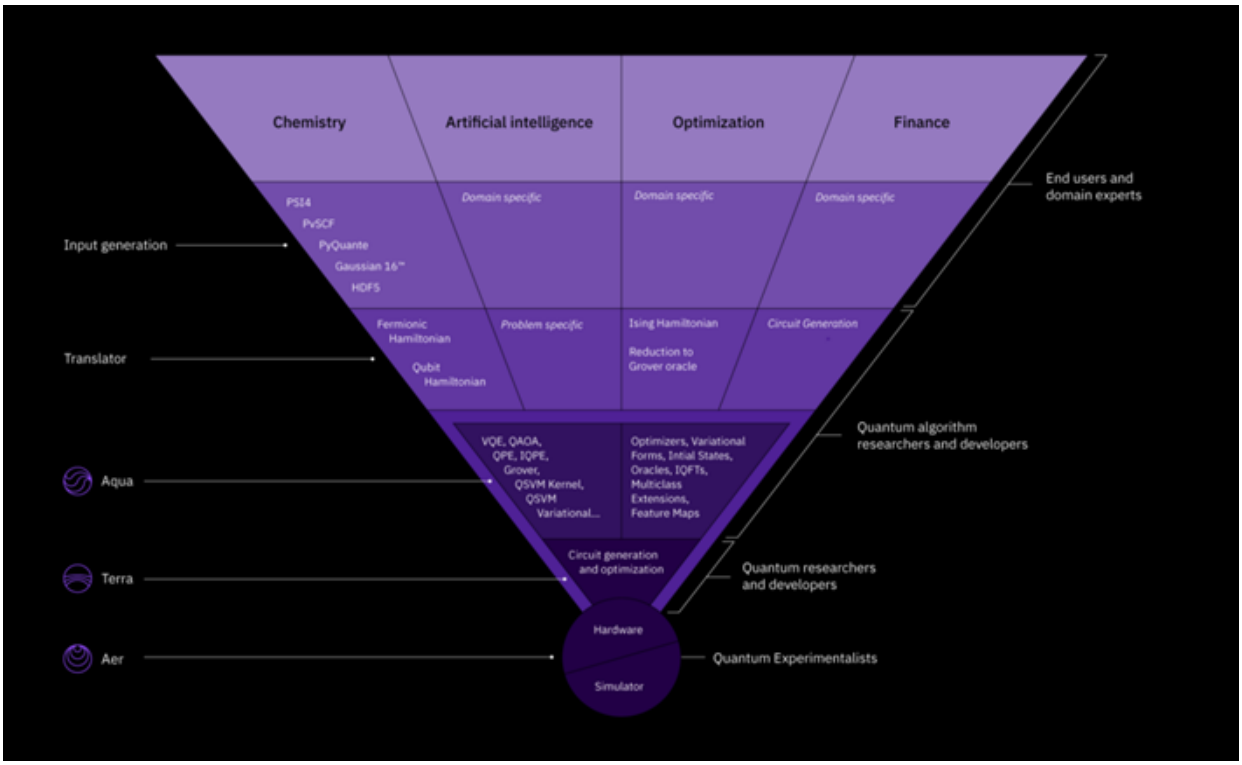


Figura 2.2. Ecosistema Qiskit. Immagine tratta da qiskit.org

E' stato da poco rilasciato il nuovo modulo di ottimizzazione in Qiskit, Qiskit Optimization [22]. Questo combina le migliori tecniche di ottimizzazione classica con algoritmi quantistici all'avanguardia per consentire la prototipazione rapida e promuovere la ricerca. Il nuovo modulo è stato sviluppato da ricercatori IBM Quantum e da collaboratori della comunità di IBM open source.

Con questa versione, il modulo Qiskit Optimization consente una modellazione semplice ed efficiente dei problemi di ottimizzazione utilizzando DOcplex - modellazione CPLEX IBM Decision Optimization. Vi è un'interfaccia uniforme e conversione automatica tra diverse rappresentazioni dei problemi. Ciò consente di risolvere i problemi utilizzando un ampio set di algoritmi.

Qiskit supporta i programmi quadratici con variabili binarie, intere e continue, nonché vincoli di uguaglianza e disuguaglianza. Questa classe di problemi di ottimizzazione ha una vasta quantità di applicazioni rilevanti e copre alcune sottoclassi molto interessanti, come programmi quadratici che possono essere risolti in modo efficiente da algoritmi di ottimizzazione classici oppure problemi di ottimizzazione

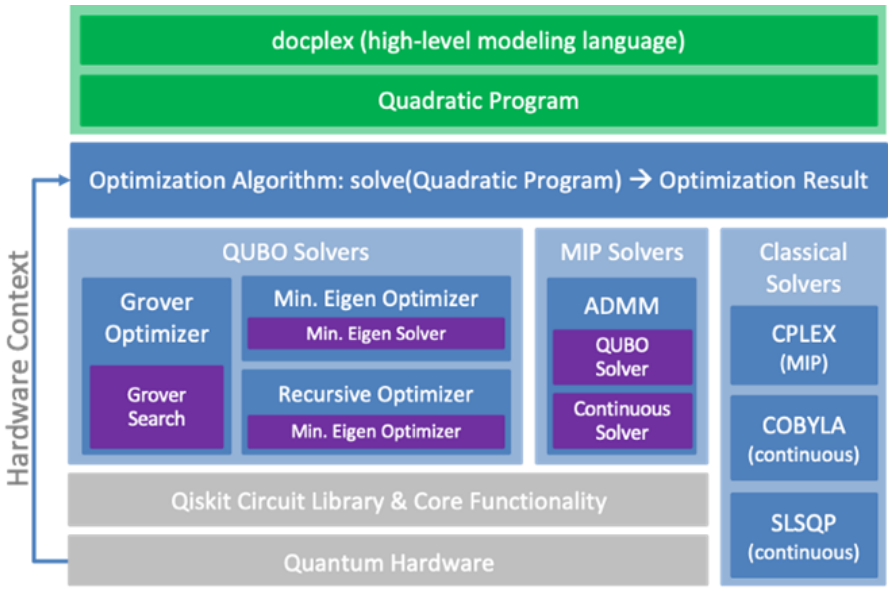


Figura 2.3. Quantum Optimization in Qiskit. Immagine tratta da qiskit.org

Quadratic Unconstrained Binary Optimization (QUBO), che coprono molti problemi NP-completi, cioè classicamente intrattabili.
Risolvere un QUBO equivale a trovare uno stato fondamentale di una corrispondente Hamiltoniana di Ising e molti algoritmi di ottimizzazione si basano su questa procedura di risoluzione. Pertanto, Qiskit utilizza degli algoritmi specifici per calcolare gli stati fondamentali, come Variational Quantum Eigensolver (VQE e QAOA). Per sfruttare questi algoritmi fondamentali, il modulo di ottimizzazione Qiskit fornisce il Minimum Eigen Optimizer.
Il Minimum Eigen Optimizer traduce un determinato programma quadratico appropriato in un hamiltoniano, applicando tutte le conversioni necessarie, chiama un determinato Eigensolver minimo e restituisce i risultati di ottimizzazione.

Sebbene la maggior parte degli algoritmi quantistici per QUBO siano correlati alla ricerca di stati fondamentali di un hamiltoniano, questa non è l'unica possibilità.
I collaboratori di JPMorgan Chase hanno sviluppato un modo efficiente che consente l'ottimizzazione basata su Grover Adaptive Search (GAS), con una velocità quadratica rispetto alla ricerca di forza bruta.
Dato il design unificato, è semplice risolvere un dato QUBO utilizzando il formalismo hamiltoniano o l'ottimizzatore di Grover, senza dover cambiare il modello.

Nel quarto capitolo viene spiegato come funzionano gli algoritmi di risoluzione di un problema e quali prestazioni offrono.

Capitolo 3

Problema e soluzione proposta

3.1 Problemi di ottimizzazione

In termini generali, dato uno spazio X (spazio delle variabili), un insieme $S \subseteq X$ e una funzione $f : S \to R$, un problema di Ottimizzazione (in forma di minimizzazione) può essere formulato come

$$\begin{cases} minf(x) \\ x \in S \end{cases} \tag{3.1}$$

e consiste nel determinare, se esiste, un punto di minimo della funzione f tra i punti dell'insieme S, ovvero un punto $x* \in S$ tale che $f(x*) \leq f(x)$ per ogni $x \in S$. Analogamente un problema di Ottimizzazione (in forma di massimizzazione) puà essere formulato come

$$\begin{cases} maxf(x) \\ x \in S \end{cases} \tag{3.2}$$

e consiste nel determinare, se esiste, un punto di massimo della funzione f tra i punti dell'insieme S, ovvero un punto $x* \in S$ tale che $f(x*) \geq f(x)$ per ogni $x \in S$. Si parla indifferentemente di problemi di massimo o di minimo in quanto vale

$$minf(x) = -max(-f(x)), x \in S \tag{3.3}$$

La funzione f viene chiamata funzione obiettivo e l'insieme S insieme ammissibile cioè l'insieme delle possibili soluzioni del problema.
L'insieme ammissibile è solitamente espresso da un numero finito di relazioni di uguaglianza o disuguaglianza.

Considerando formalmente le funzioni $g_i : Rn \to R, i = 1, ..., m$, un insieme ammissibile definito da vincoli di disuguaglianza è così descritto:

$$S = \{x \in R^n | g_1(x) \leq 0, g_2(x) \leq 0, ..., g_m(x) \leq 0\} \tag{3.4}$$

Ogni disuguaglianza $g_i(x) \leq 0$ è chiamata vincolo e l'insieme ammissibile è costituito da punti che risolvono il sistema di disuguaglianze non lineari

$$\begin{aligned} g_1(x) &\leq 0 \\ g_2(x) &\leq 0 \\ g_3(x) &\leq 0 \\ &... \\ g_m(x) &\leq 0 \end{aligned}$$

Un problema generico, con vincoli di uguaglianza e disuguaglianza, ha quindi la seguente forma:

$$\begin{aligned} &minf(x) \\ &g(x) \leq 0, \\ &h(x) = 0 \end{aligned}$$

3.2 Problema di ottimizzazione applicato ad un caso reale

3.2.1 Descrizione del problema preso in analisi

Il problema che si vuole risolvere è un problema di ottimizzazione legato alle linee di trasporto. Si prendono in considerazione le linee gtt della città di Torino.

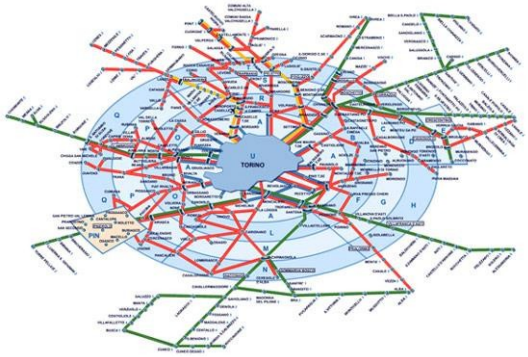


Figura 3.1. Visualizzazione delle linee di trasporto. Immagine tratta da www.sfmtorino.it

Sono oggetto di studio le linee dei bus e tram.
Si consideri ad esempio una linea qualsiasi gtt.
Di seguito vi è un immagine che mostra una corsa della linea 10.

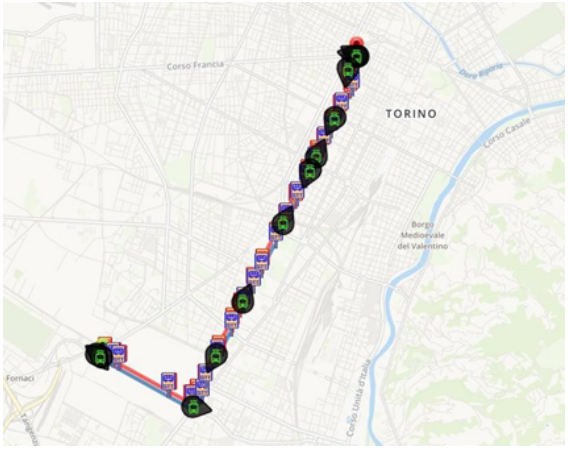


Figura 3.2. Corsa della linea 10. Immagine tratta da www.gtt.to.it

La linea 10 esegue il tragitto mostrato in figura 3.2.
Il bus esegue questo tragitto più volte al giorno, contiene quindi diverse corse.
Ogni corsa ha un orario di partenza, inotre vi sono degli stop in cui il bus si ferma.
Ad ogni stop arriva ad orario x e riparte ad un orario y.

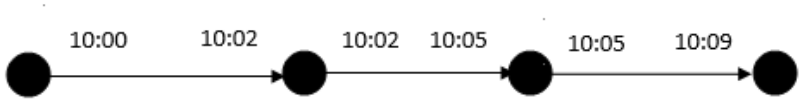


Figura 3.3. Esempio di tragitto

Di seguito viene rappresentato in maniera molto semplice una parte del tragitto di una corsa della linea 10. Ogni stop ha un identificativo, un orario di arrivo e di partenza per ogni corsa.
I dati relativi agli orari, agli stop e alle corse per linea sono i dati di input del problema e sono descritti in tabelle.
Come la linea 10 lo stesso vale per tutte le altre linee della città.

Il problema è un problema di massimizzazione delle interconnessioni.
Per interconnessione si intende uno stop, che interconnette due o più linee di trasporto.

Figura 3.4. Interconnessione

Per massimizzazione delle interconnessioni si intende la ricerca di un massimo, basandoci su alcuni dati ricevuti in input e spostando gli orari di partenza delle corse delle linee, in avanti o indietro, di alcuni minuti.

Si prenda, ad esempio, lo schema precedente relativo alla corsa 1 della linea 10 e si sposti l'orario di inizio corsa in avanti di 2, di 3 e di 5 minuti (Figura 3.5,3.6,3.7).
Per il problema ogni stop ha una priorità, e questo dato viene dato in input.

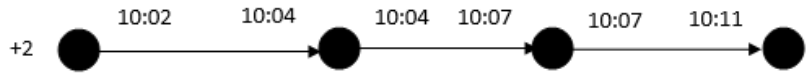


Figura 3.5. Esempio di tragitto con applicazione scostamento di 0 minuti

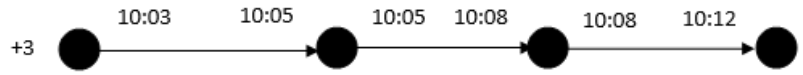


Figura 3.6. Esempio di tragitto con applicazione scostamento di 2 minuti

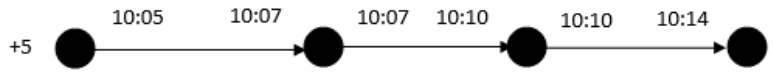


Figura 3.7. Esempio di tragitto con applicazione scostamento di 3 minuti

Lo scopo del problema è: capire di quanto spostare la linea 10 e, di conseguenza, anche tutte le altre linee del problema, in modo da ottenere un valore massimo di priorità-stop.
Inoltre, c'è da considerare anche il tempo di attesa.
Ad esempio, un passeggero scende dalla linea 10 allo stop 1 e vuole salire sulla linea 16 (Figura 3.8).

Figura 3.8. Interconnessione di due linee

In questo caso il passeggero scende dal 10 alle 10:17, attende 6 minuti che arrivi la linea 16, e sale sul 16. Il tempo di attesa è 6 minuti.
Il tempo di attesa deve essere calcolato in questo caso come per tutti gli altri casi, e deve sempre essere minore di un massimo che ci viene dato in input.

In particolare, il programma deve riuscire anche a calcolare il tempo di attesa minimo che si ottiene considerando due linee che si incontrano in uno stop e i relativi spostamenti.

In output, quindi, si dovrà ottenere il valore delle variabili binarie che indicano di quanto spostare ogni corsa di ogni linea, in modo da avere massimo valore di priorità, ed inoltre il valore della variabili binarie che indicano quale corse della linea successiva è la più adatta, dato il tempo di arrivo in un determinato stop, per ottenere minimo valore di tempo di attesa.

Nel caso preso in analisi si considereranno:

- 3 linee: Linea1, Linea2, Linea3
- 2 corse giornaliere per ogni linea: corsa1, corsa2
- 2 stop: stop1, stop2
- 3 scostamenti: 0, 2, 3 minuti per ogi corsa

In particolare si calcolerà:

- Massimo valore priorità
- Lo scostamento ideale per ogni corsa delle 3 linee.
- Il minor tempo di attesa totale considerando tutte le combinazioni possibili tra le corse.
- La corsa ideale da prendere in base allo stop, all'orario di arrivo e alla linea successiva.

Molto importante risulta essere l'impostazione del problema. Per i motivi che vengono spiegati nel quarto capitolo, il problema avrà:

- Variabili binarie da calcolare e che rappresenteranno l'output del problema.
- Vincoli che il programma dovrà rispettare. I vincoli impongono determinate condizioni sulle variabili.
- Funzione di massimizzazione quadratica attraverso cui si calcolerà il massimo.

3.2.2 Soluzione proposta

Sono state implementate diverse strutture dati. In questo paragrafo vengono spiegate ed illustrate. Verrà inoltre descritto il modello implementato per la risoluzione del problema.

- Strutture dati

  1. Vettore AllStop

Stop1	Stop2	...	StopN

     Il vettore contiene in ogni cella le matrici Stop, risulta essere quindi un vettore di matrici.
     In questo codice verranno considerati, per semplicità, solo 2 stop, quindi avremo due matrici Stop1 e Stop2.
  2. Matrici Stop{x}
     La matrice Stop è descritta nella tabella seguente.

	Corsa1	Corsa2	CorsaN	Corsa1
Linea1	10:00	10:20	...	10:00
Linea2	10:17	10:24	...	10:17
Linea3	10:12	10:17	...	10:12
...				
LineaN	10:22	10:26	...	10:22

     La matrice contiene l'orario di arrivo di una determinata corsa di una linea di trasporto allo stop considerato; in particolare le righe identificano le linee di trasporto e le colonne le corse corrispondenti.

     Importante risulta essere l'ultima colonna della tabella, che fa riferimento alla prima corsa giornaliera della corrispondente linea.
     Viene riportata la prima corsa giornaliera, nell'ultima colonna, in quanto necessaria per calcolare il tempo di attesa nel caso in cui la linea che si vuole prendere non abbia più corse nella stessa giornata e quindi si è costretti ad aspettare la prima corsa giornaliera del giorno successivo.

     Esempio: Un passeggero scende dalla linea 1 alle 11:55 e vuole prendere la linea 2, la cui prima corsa giornaliera sarà alle 12:25 del giorno dopo.

     Nel caso preso in analisi si considerano solo 2 stop, quindi si avranno due matrici Stop1 e Stop2.

Viene riportato il codice che istanzia le matrici Stop1, Stop2 e il vettore AllStop.

```
AllStop= []

Stop1 = array([[1000, 1020, 3400],
               [1017, 1024, 3417],
              [1012,   1017, 3412]])

Stop2 = array([[1005, 1025,  3405],
               [1019, 1026, 3419],
              [1020,   1025, 3420]])

AllStop.append(Stop1)
AllStop.append(Stop2)
```

Gli orari sono riportati in numeri a 4 cifre per semplicità. Quindi 10:00 verrà indicato dal valore 1000.

Come si può notare, inoltre, l'ultima colonna della tabella ha un valore diverso. Quest'ultima si riferisce alla prima corsa giornaliera del giorno, tuttavia affinche sia possibile calcolare il tempo di attesa, che deve essere un numero positivo, si considera l'orario effettivo con l'aggiunta di 2400, che rappresenta 24:00 e quindi 24 ore.
Questo valore indica le ore presenti in un giorno e vengono aggiunte per indicare la prima corsa del giorno dopo in modo che il tempo di attesa risulti sempre positivo.

3. Vettore stop_priority

PriorityStop1	PriorityStop2	...	PriorityStopN

Questo vettore indica la priorità di ogni stop. Ogni stop ha una priorità, questo dato viene dato in input. Nel caso preso in analisi vi sono solo 2 stop, quindi 2 valori di priorità.

4. Vettore corse_per_linea

CorseLinea1	CorseLinea2	...	CorseLineaN

Corse_per_linea indica il numero di corse per ogni linea. Ogni linea ha un numero prefissato di corse e questo dato viene dato in input. Nel caso preso in analisi vi sono solo 3 linee ognuna delle quali ha 2 corse giornaliere.

```
stop_priority=array([2,5])

corse_per_linea=array([2,2,2])
```

5. Vettori linea{x}

PassaPerStop1	PassaPerStop2	...	PassaPerStopN

I vettori linea hanno una struttura così indicata.
In ogni cella vi è il valore 1 o 0 e sta ad indicare se la linea presa in analisi passa o no per un determinato stop. Ogni linea passa per determinati stop e questo dato viene dato in input.
Nel caso preso in analisi vi sono solo 2 stop e le tre linee passano per questi due stop quindi avranno una struttura del genere:
Linea1

1	1	1

Linea2

1	1	1

Linea3

1	1	1

6. Vettore linee

VettoreLinea1	VettoreLinea2	...	VettoreLineaN

Il vettore linee è un vettore di vettori.
In ogni cella c'è il vettore linea corrispondete all'indice in cui si trova.
In questo caso il vettore linee avrà solo 3 celle.

```
linee= []

linea1 = array([1,1]) # in questo caso tutte e tre le
   linee passano per entrambi gli stop
linea2 = array([1,1])
linea3 = array([1,1])

linee.append(linea1) # si concatena
linee.append(linea2)
linee.append(linea3)
```

7. Vettori stop{x}_pos

PosizioneLinea1	PosizioneLinea2	...	PosizioneLineaN

I vettori stop_pos ci indicano, in baso allo stop a cui si riferiscono, in quale riga della matrice Stop si trovano le linee di trasporto. Nel caso in cui una linea non passi per lo stop a cui il vettore stop_pos fa riferimetno, nella cella corrispondente vi sarà un -1.

In questo caso le 3 linee passano per i 2 stop, in particolare sia nella matrice Stop1 che nella matrice Stop2 la prima riga si riferisce alla linea 1, quindi la essa avrà posizione 0 in entrambe le matrici. Lo stesso vale per la linea 2 che avrà posizione 1 e la linea 3 che avrà posizione 2.

Stop1_pos

0	1	2

Stop2_pos

0	1	2

Queste strutture dati sono utili nel momento in cui si lavora con molte linee e stop e non si vuole creare delle matrici Stop che abbiano molte righe vuote. Così si garantisce che le matrici Stop contengano solo le linee che passano per lo stop considerato.

```
stop_pos=[]

stop1_pos= array([0,1,2]) #la linea 1 ha posizione 0,
#la linea 2 ha posizione 1 e la linea 3 ha posizione 2
#nella matrice Stop1

stop2_pos= array([0,1,2])

stop_pos.append(stop1_pos)
stop_pos.append(stop2_pos)
```

8. Vettore stop_pos

stop_pos1	stop_pos2	...	stop_posN

Il vettore stop_pos contiene tutti i vettori stopx_pos, risulta quindi essere un vettore di vettori.

9. Vettore T

Tempo1	Tempo2	...	TempoN

T è il vettore degli scostamenti possibili.
Gli scostamenti nel tempo possono avvenire in avanti o indietro. In questo caso si effettua uno scostamento di 0, 2 o 3 minuti.
T

0	2	3

```
T=array([0,2,3])

```

10. Matrici Scostamento{x}

	T[0]	T[1]	T[...]	T[N]
corsa1	0	1	...	0
corsa2	1	0	...	0
corsa3	0	1	...	0

La matrice Scostamento1 corrisponde alla linea 1. Indica per ogni corsa della linea 1 qual è lo scostamento relativo.

Ci sono 3 corse siccome la terza è la prima corsa giornaliera, ai cui orari di arrivo nei vari stop è stato aggiunto 24.
La spiegazione è stata già riportata in precedenza, ricapitolando potrebbe essere più chiaro usare un esempio.
Si scende da una corsa della linea 1 e si vuole salire su una corsa della linea 2, tuttavia tutte le corse giornaliere della linea 2 sono già passate. Si deve quindi attendere la prima corsa giornaliere della linea 2, del giorno dopo. Si consideri che la corsa in questione arrivi alle 2:00 del mattino,tuttavia pero si è scesi dalla corsa della linea 1 alle 23:30, si deve attendere quindi 2 ore e mezza.
Se si esegue la differenza fra 2:00 e 23:30 mi ritrovo un tempo di attesa negativo. Di conseguenza, si aggiunge 24 a 2:00, per indicare che si tratta di una corsa del giorno dopo, ed otteniamo un tempo di attesa positivo.
Verrà aggiunto un vincolo che forzerà lo stesso scostamento per la prima e l'ultima riga di Scostamenti, in quanto si tratta di fatto della stessa corsa.
Nell'esempio preso in considerazione ci sono 3 scostamenti, quindi tre colonne (0, 2 e 3 minuti).

La posizione dell'1 indica quale scostamento prendere.
Considerando la seguente matrice relativa alla linea 1, si avrà:

Scostamenti1

0	1	0
0	0	1
1	0	0

Si ha che la prima corsa ha uno scostamento di 2 minuti, la seconda di 3 minuti e la terza di 3 minuti.
I valori all'interno di ogni matrice verranno calcolati dalla funzione di massimizzazione, e ci indicano lo scostamento ideale per ogni corsa di ogni linea.

11. Vettore Scostamenti

Scostamento1	Scostamento2	...	ScostamentoN

Il vettore Scostamenti è un vettore di matrici che contiene quindi tutte le matrici Scostamenti che nel caso considerato sono tre.

```
Scostamenti = []
idx = [(i, j) for i in range(0,3) for j in range(0,3)]
#considerando 2 corse per linea, piu' quella del giorno
    dopo e considerando 3 scostamenti ( 0, 2 o 3)

for y in range(0,3): # cosiderando 3 linee
        Scostamenti.append(mdl.binary_var_dict(idx, name="
    x"+str(y)))
```

12. Intersezioni

	Linea1	Linea2	Linea3	...	LineaN
Linea1					
Linea2					
Linea3					
...					
LineaN					

La matrice intersezioni serve per capire le linee dove si intersecano.
In ogni cella vi sarà il vettore che contiene gli stop in cui la linea indicata dalla colonna si interseca con la linea indicata dalla riga di questa matrice.
Ad esempio nel caso preso in esempio si ha che la linea 1 e 2 si intersecano nello stop1 e stop2 , quindi Intersezioni[0,0] avrà:

1	2

```
intersezioni=zeros((3,3),dtype=object)

```

Questa matrice verrà popolata man mano dal programma ed è una matrice di vettori.

13. Vettore A e B
I vettori A e B servono per memorizzare dei dati importanti che verranno usati nei vincoli. In particolare, in A vi saranno i tempi di attesa che il programma calcolerà per tutte le corse che si incontrano.

    Tempo di attesa = orario di partenza - orario di arrivo

    Orario di arrivo = orario iniziale di arrivo + scostamento

    Orario di partenza = orario iniziale di partenza + scostamento

    Esempio:
    Una pesseggero arriva allo stop 1, scendendo dalla seconda corsa della linea 1, e dovrà prendere la prima corsa che arriva della linea 2.
    L'orario di arrivo è calcolato selezionando AllStop[1], che restituirà la matrice Stop1, quindi si seleziona Stop1[0,1] che restituirà l'orario di arrivo della seconda corsa della linea 1 allo stop 1. A questo valore va aggiunto lo scostamento calcolato del codice, che si troverà nella cella con valore uno presente in Scostamento1[0], per poi ottenere l'orario di arrivo definitivo.
    Per l'orario di partenza il procedimento è lo stesso.

    La differenza fra orario di partenza e orario di arrivo determina il tempo di attesa, che deve essere in qualsiasi caso un tempo espresso in minuti, positivo.
    Verranno memorizzati tutti i tempi di attesa, calcolati tra le varie corse, nella struttura dati A. In seguito vi sarà un vincolo applicato su A per far sì che i tempi di attesa siano positivi.

    B è una struttura dati che memorizza anch'essa i tempi di attesa, però considerando alcune specifiche.

    Ciò che memorizza B verrà dato in pasto alla funzione di massimizzazione, quindi B non potrà contenere i tempi di attesa, in quanto il calcolo

dei tempi di attesa è un problema di minimo. Nel caso preso in analisi si vuole implementare un problema di massimizzazione.
Si suppone che ci sia un tempo massimo, espresso in minuti, che un utente può attendere. Ad esempio si suppone un tempo di 40 minuti, al quale verranno sottratti i tempi di attesa per calcolare il tempo di attesa minimo in un problema di massimizzazione.

Esempio:
Passeggero che scende dalla linea 1 e deve salire sulla linea 2, si suppone il caso in cui salga sulla prima corsa della linea 2 e debba aspettare 3 minuti, mentre nel caso in cui salga sulla seconda corsa aspetterà 6 minuti. Si vuole considerare migliore la prima opzione in quanto il passeggero attende di meno.
Applicando un problema di massimo tra (40-3) e (40-6), il programma selezionerà come valore più alto quello risultante dalla prima sottrazione, quindi considererà il tempo di attesa minimo, che di fatti risulta essere 3 minuti e non sei.

Nel vettore A si avrà il valore ottenuto da (tempo massimo di attesa (40 - tempo di attesa) , per tutte le combinazioni delle corse delle linee prese in considerazione.

Inoltre, siccome in input viene dato un vettore che indica la priorità di uno stop, si moltiplicherà questo valori per (40 – tempo di attesa), e si darà quest'espressione in pasto alla funzione di massimizzazione.

Vettore A

(40 - tempo attesa)
...
...

Vettore B

Stop_priority[i]*(40 - tempo attesa)
...
...

Il vettore B, come il vettore A, verranno popolati man mano che il codice è in esecuzione, e conterranno i dati di tutte le corse che si incontrano.
Quindi in ogni cella del vettore A e B si fa riferimento al tempo di attesa per una specifica corsa di arrivo e una specifica corsa di partenza, in uno specifico stop.

14. Matrice
    Matrice è una struttura dati complessa, è una matrice di matrici, che contiene i tempi di attesa che il programma calcolà.

	Linea1	Linea2	Linea3	...	LineaN
Linea1					
Linea2					
Linea3					
...					
LineaN					

    Matrice è simile a Intersezioni, tuttavia la seconda struttura contiene vettori che indicano gli stop in cui la linea indicata dalla riga e quella indicata dalla colonna si intersecano. Matrice, invece, contiene delle matrici così descritte:

	corsa1	corsa2	...	corsaN
stop1				
stop2				
...				
stopN				

    Prendendo in considerazione la cella [0,0] di Matrice, essa fa riferimento all'intersezione tra linea 1 e 2. In particolare, troveremo una matrice strutturata come mostrato sopra, dove nella cella [0,0] vi sarà il tempo di attesa calcolato per un passeggero che scende dalla prima corsa della linea 1 allo stop 1, e deve salire sulla linea 2. Mentre nella cella [0,1] vi sarà il tempo di attesa calcolato per chi scende dalla seconda corsa della linea 1 allo stop 1, e deve salire sulla linea 2.

```
1 Matrice = zeros((3,3),dtype=object)
```

    Questa matrice verrà popolata durante l'esecuzione, in quanto i tempi di attesa dipendono da quanto scostamento viene calcolato per la corsa di arrivo e di partenza. Quindi non può essere popolata a priori in quanto non si conosce a priori gli scostamenti ottimi.

    In dettaglio succede questo, si passano ad una funzione il numero della corsa e della linea di arrivo, il numero della linea di partenza, e lo stop. Questa funzione calcola l'ipotetico tempo di attesa, e lo inserisce nel vettore A, in particolare nell'ultima cella di A tramite il metodo append. (Figura 3.9)

    A e B non hanno una dimensione iniziale , ma in base alla quantità di dati che si sta manipolando, viene incrementata o decrementata la loro dimensione.

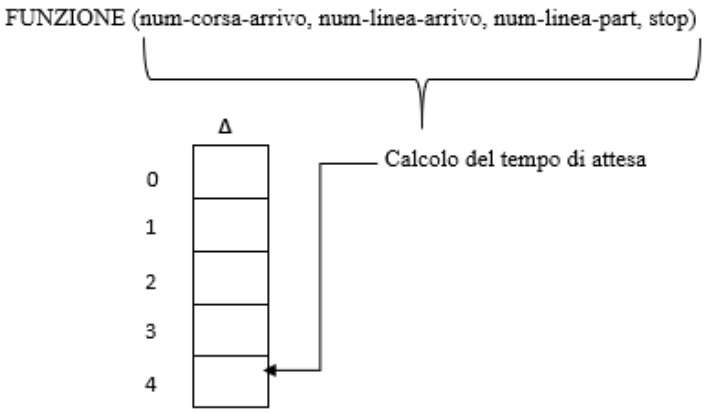


Figura 3.9. Inserimento valore calcolato dalla funzione in A

Il vettore A non contiene altre informazioni, ma solo il tempo di attesa. Per sapere quali corse e linee si sta considerando, inseriamo l'indice di A nella posizione specifica di Matrice.

Il perché si usano A, B e Matrice e non solo Matrice, viene spiegato in seguito.

15. Nextc
    Nextc risulta essere l'ultima struttura dati.
    Finora sono stati considerati i tempi di attesa, data una corsa di una linea in arrivo, uno stop, e una linea di partenza. Tuttavia come è possibile capire quale deve essere la corsa della linea di partenza su cui l'utente deve salire?
    Per il calcolo del tempo di attesa è necessario identificare qualce corsa della linea da prendere si vuole considerare.

    Il passeggero scende dalla prima corsa della linea 1 allo stop 1, deve salire sulla linea 2. In particolare qual è la corsa della linea 2 che passerà per prima e che quindi garantisce un tempo di attesa minimo?
    Potrebbe essere la prima corsa, come potrebbe essere la quarta o la decima. Dipende dagli orari specificati nelle matrici degli Stop e dagli scostamenti calcolati per le corse.

    Introduciamo quindi la struttura dati nextc.
    È un vettore di vettori, quindi in pratica una matrice, ed è binario. Per ogni riga ci sarà solo una cella con valore 1, tutte le altre saranno settate a 0, e indica quale corsa della linea in cui l'utente vuole salire è la più

adatta considerando il tempo di arrivo in uno stop.

Esempio:
Il passeggero arriva alle 10:00 allo stop 1 scendendo dalla prima corsa della linea 1. L'orario di arrivo è quindi 10:00, l'orario di partenza dipende da quale corsa della linea 2 verrà considerata.

Si supponga che la prima corsa della linea 2 arrivi alle 9:00, la seconda alle 9:40, la terza alle 10:10 e la quarta alle 10:30. La corsa più adatta è la terza, in quanto ha un orario successivo all'orario di arrivo, quindi un tempo di attesa positivo, ed inoltre garantisce un'attesa minore rispetto alla quarta corsa.

Ricapitolando:

- orario di arrivo alle 10:00 dalla prima corsa della linea 1
- stop 1
- i vuole prendere la linea 2, quale corsa è la più adatta?

Considerando nel caso preso in analisi tre corse per la linea 2 si avranno tre possibili casi.

ORARIO DI PARTENZA PRIMA CORSA DELLA LINEA 2 – ORARIO DI ARRIVO	ORARIO DI PARTENZA SECONDA CORSA DELLA LINEA 2 – ORARIO DI ARRIVO	ORARIO DI PARTENZA TERZA CORSA DELLA LINEA 2 – ORARIO DI ARRIVO

Si fissa il tempo di arrivo e si calcolano i tempi di partenza per tutte le corse della linea 2. La differenza fra i due sono i tempi di attesa.

Il vettore indicato in alto, i cui valori sono sottratti a 40, si trovano in ogni cella di A. Quindi una cella di A non contiene più un singolo valore, ma un vettore.
In ogni riga l'orario di arrivo è uguale per tutte le colonne. Le colonne fanno riferimento agli orari di partenza delle corse della linea che il passeggero vuole prendere. Una riga di A avrà questa struttura

Guardando la figura 3.10 risulta essere più chiaro l'utilizzo di nextc. Contiene dei vettori binari, in cui solo una cella tra tutte nella stessa riga avrà valore 1, quindi indica quale corsa della linea da prendere è la più adatta.

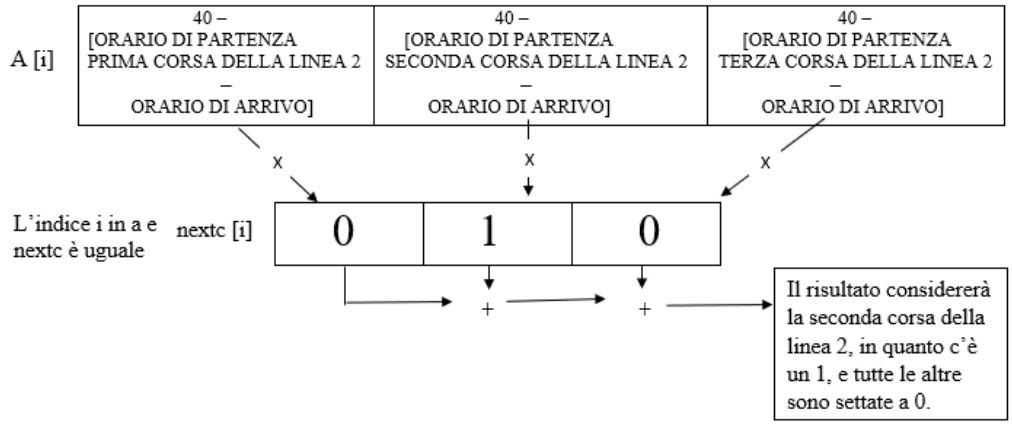


Figura 3.10. Moltiplicazione tra A e nextc

L'algoritmo calcolerà nextc, che contiene tanti vettori come quello raffigurato, in numero uguale a quanti ne contiene il vettore A, e quindi anche a quanti ne contiene il vettore B.
Si impone un vincolo che setterà solo una cella per ogni riga di nextc a 1, e tutte le altre a 0. La struttura A e B è mostrata in figura 3.11 e 3.12

Viene spiegato quindi il perché non può inserire ciò che contiene A dentro Matrix. Quest'ultima sarebbe stata una struttura troppo complessa, una matrice di matrici di vettori.
In ogni cella di Matrix ci sarà solo l'indice di A corrispondente, e quindi anche di nextc, in modo che quando il computer calcolerà gli scostamenti ottimi e popolerà nextc con opportuni uni e zeri, verrà prelevato il valore ideale dato l'indice.
B contiene le stesse informazioni di A considerando le priorità.

Tutte le righe di B, moltiplicate per le apposite righe di nextc, vengono date in pasto alla funzione di massimizzazione. Si guardi la figura 3.17 per capire meglio.

Inserendo nella funzione di massimizzazione tutte le righe di B moltiplicate per le apposite righe di nextc, il computer calcolerà sia gli scostamenti ottimi, all'interno della struttura Scostamenti, sia i vettori binari in nextc.
In sostanza si hanno due struttura dati binarie che verranno calcolate nello stesso momento, la struttura Scostamenti, che fa riferimento a quanto scostare ogni corsa di ogni linea e serve per calcolare i tempi di attesa presenti in A e B, e la struttura nextc che indica quale corsa della linea in cui su vuole salire è la più adatta, considerando i vincoli.

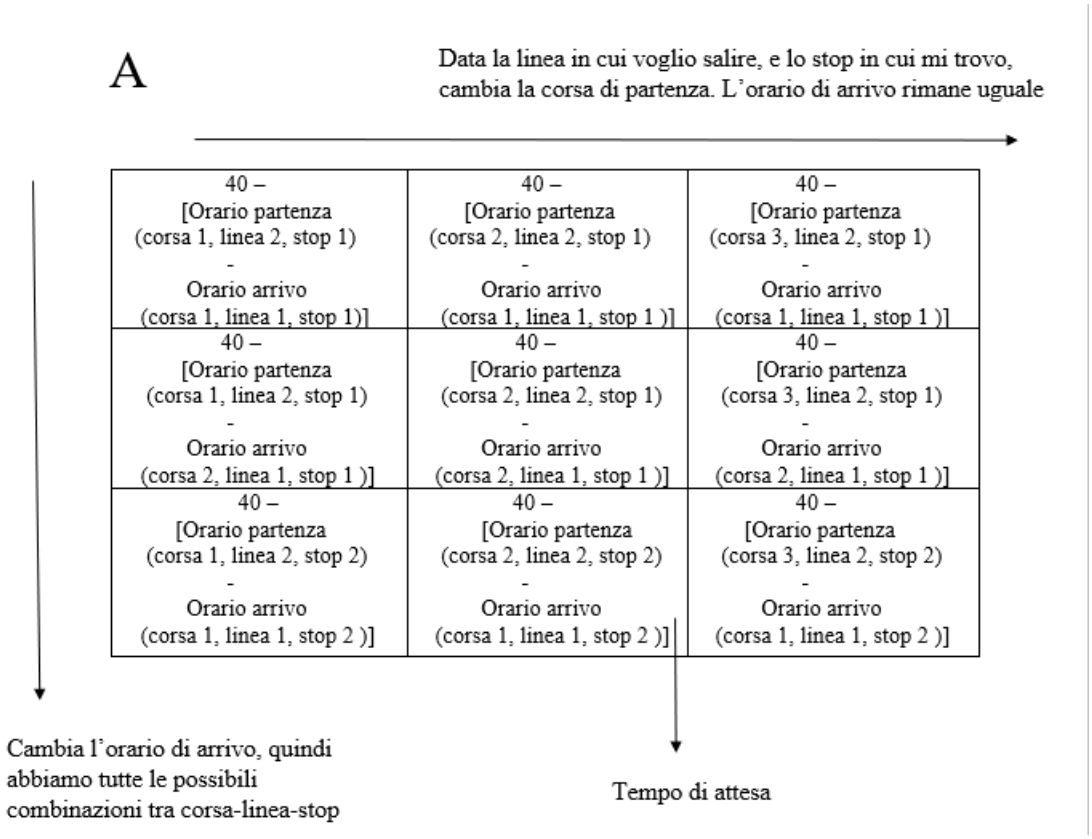


Figura 3.11. Struttura Matrice A

B

STOP_PRIORITY [0] x A [0, 0]	STOP_PRIORITY [0] x A [0, 1]	STOP_PRIORITY [0] x A [0, 2]
STOP_PRIORITY [0] x A [1, 0]	STOP_PRIORITY [0] x A [1, 1]	STOP_PRIORITY [0] x A [1, 2]
STOP_PRIORITY [1] x A [2, 0]	STOP_PRIORITY [1] x A [2, 1]	STOP_PRIORITY [1] x A [2, 3]

Figura 3.12. Struttura Matrice B

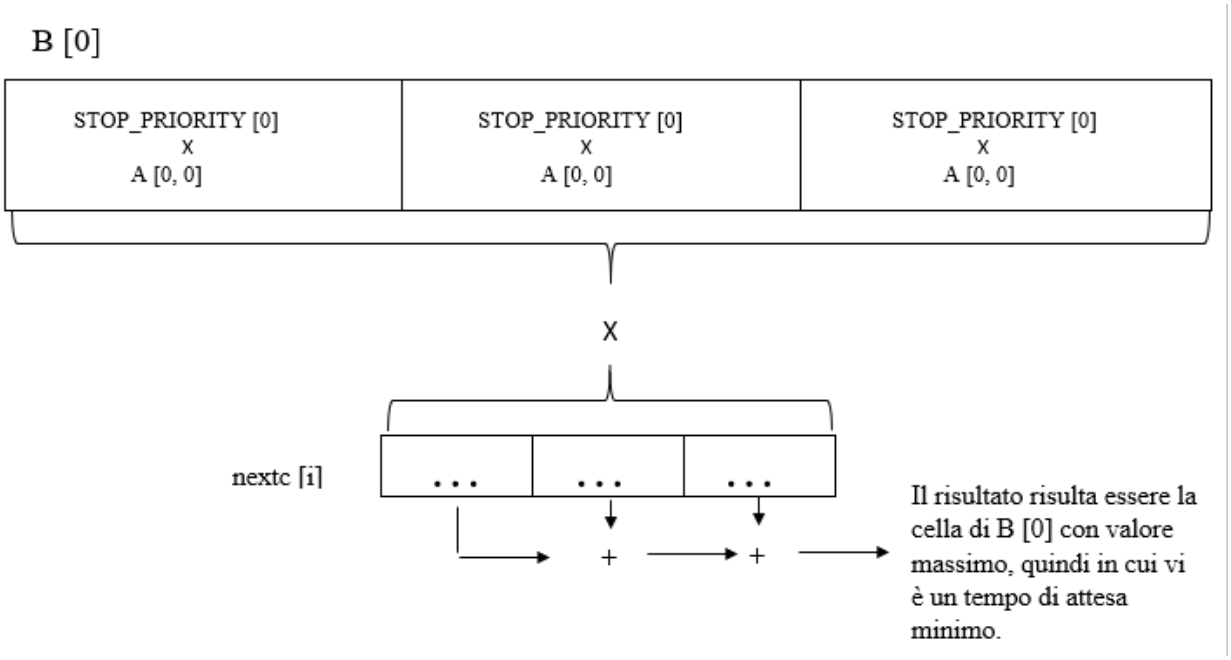


Figura 3.13. Moltiplicazione tra B e nextc

- Vincoli

  1. Nella struttura dati Scostamenti, ogni riga di ogni matrice ha tutti valori a 0 tranne uno settato a 1.

```
for x in range(0,len(linee)):
   for j in range(0,corse_per_linea[x]+1):
      mdl.add_constraint(mdl.sum(Scostamenti[x][j,k] for k
    in range(0,Nscostamenti))==1)  # si definisce che solo
    un valore di Scostamenti[x][j] puo' avere valore 1, e
   indichera' quale scostamento prendere in base alla
   posizione dell'1

```

  Si ricorda che in ogni cella di scostamenti si fa riferimento a una linea specifica. La prima cella, ad esempio, si riferisce alla linea 1. In ogni cella vi è una matrice e ogni riga della matrice è un vettore binario, che indica di quanto spostare la corsa corrispondente alla riga. Si veda figura 3.14

  Il vincolo raffigurato indica che, per tutte le celle in Scostamenti, che in questo caso sono 3, Per tutte le righe delle matrici Scostamento, che in questo caso sono uguali a corse_per_linea + 1, siccome in sostanza vi sono 2 corse, ma la terza è la prima con l'aggiunta di 24, e per tutte le

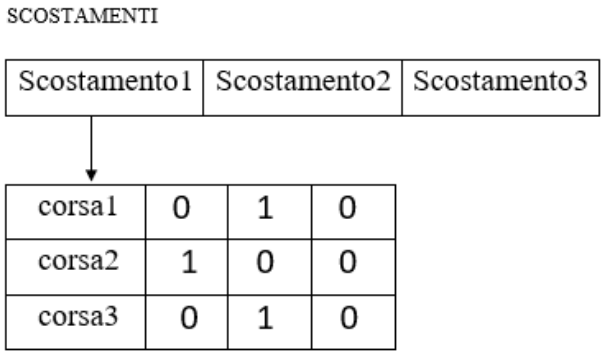


Figura 3.14. Struttura del vettore Scostamenti

colonne nelle matrici Scostamenti, che in questo caso sono 3 in quanto vengono considerati 3 scostamenti, la sommatoria delle celle deve essere uguale a 1. Quindi si avrà solo una cella settata a 1 e tutte le altre a 0.

2. La prima e l'ultima corsa in tutte le matrici Scostamento interne a Scostamenti, devono avere lo stesso valore, siccome, si tratta semplicemente della stessa corsa.

```
for x in range(0,len(linee)):
    for k in range(0,Nscostamenti): #num scostamenti 3
        mdl.add_constraint(Scostamenti[x][0,k]==
    Scostamenti[x][corse_per_linea[x],k]) # lo scostamento
    della prima corsa deve essere uguale allo scostamento
    dell'ultima, in quanto l' ultima corsa e' in realta' la
     prima corsa giornaliera, a cui viene sommato 2400 per
    riuscire a calcolare il tempo di attesa.

```

3. Per tutte le righe di nextc, vi dovrà essere solo una cella settata a 1, e tutte le altre a 0.

```
for i in range(0,len(nextc)):
    mdl.add_constraint(mdl.sum(nextc[i][k] for k in range
    (0,len(nextc[i])))==1) # si definisce che solo 1 valore
     nel vettore binario contenuto in nextc[i] puo' avere
    valore 1. Questo vale per tutti i vettori contenuti in
    nextc.
```

4. La moltiplicazione tra i vettori di A e i vettori di nextc, devono avere un valore <= 40, il che significa che il tempo di attesa dovrà essere un numero positivo; e >= 0 , il che significa che il tempo di attesa dovrà essere al massimo 40, che è il minimo che si è impostato in questo caso.

```
for i in range(0,len(A)):
    mdl.add_constraint(mdl.sum(A[i][j]*nextc[i][j] for j
   in range(0,len(nextc[i])))<=40)  # i valori contenuti
   in A[i], quindi il tempo di attesa con il 40- davanti,
   non puo' superare 40, quindi in altre parole in tempo
   di attesa effettivo non e' negativo
```

```
for i in range(0,len(A)):
    mdl.add_constraint(mdl.sum(A[i][j]*nextc[i][j] for j
   in range(0,len(nextc[i])))>=0) # questo vincolo e' il
   duale del precedente, indica che i valori contenuti in
   A[i] devono essere positivi, perche' si suppone un
   tempo di attesa massimo di 40 minuti, quindi in caso
   fossero negativi significa che il tempo di attesa
   risulta maggiore di 40, il massimo impostato.
```

5. Le corse della stessa linea devono avere un distanziamento massimo e minimo espresso in minuti

```
for i in range(0,Nstop):#num stop
    for j in range(0,len(AllStop[i])):
        for k in range(1,len(AllStop[i][j])):
            o1=(AllStop[i][j,k]//100)*60+(AllStop[i][j,k
   ]%100)
            o2=(AllStop[i][j,k-1]//100)*60+(AllStop[i][j,k
   -1]%100)
            mdl.add_constraint(((o1+sum(T[n]*Scostamenti[j
   ][k,n] for n in range(0,Nscostamenti)))-(o2+sum(T[n]*
   Scostamenti[j][k-1,n] for n in range(0,Nscostamenti))))
   >=2)
  # questo vincolo ci dice che la distanza in minuti fra
   le corse di una stessa linea deve essere compresa tra
   un minimo e un massimo. Questo vincolo e'
   personalizzabile in base al minimo-massimo per ogni
   linea
```

```
for i in range(0,Nstop):
    for j in range(0,len(AllStop[i])):
        for k in range(1,len(AllStop[i][j])):
            o1=(AllStop[i][j,k]//100)*60+(AllStop[i][j,k
   ]%100)
            o2=(AllStop[i][j,k-1]//100)*60+(AllStop[i][j,k
   -1]%100)
            mdl.add_constraint(((o1+sum(T[n]*Scostamenti[j
   ][k,n] for n in range(0,Nscostamenti)))-(o2+sum(T[n]*
   Scostamenti[j][k-1,n] for n in range(0,Nscostamenti))))
   <=40) #come il precedente indica il massimo valore che
   puo' intercorrere fra due corse della stessa linea.
```

Quindi per tutti gli stop, che in questo caso sono 2, per tutte le matrici interne in AllStop, che in questo caso sono: len (AllStop [i]),per tutte le colonne delle matrici interne Scostamento, che in questo caso sono len (AllStop[i][j]), l'orario di arrivo per la corsa k della linea j, nello stop i – l'orario di arrivo per la corsa k-1 (la precedente), della linea j, nello stop i, deve essere >= un minimo e <= un massimo dati in input.

Si ricorda che lo scostamento calcolato dal computer è dato da:

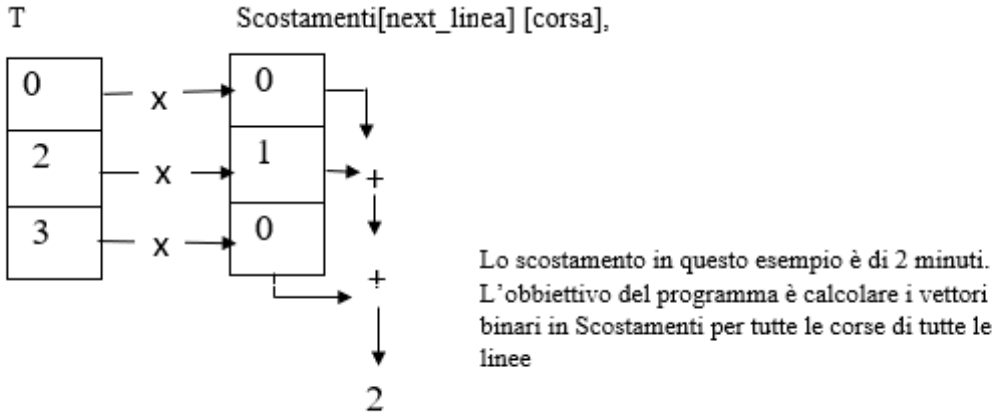


Figura 3.15. Cacolo dello scostamento

- Funzioni

1. Funzione di massimizzazione

```
max_vars_func = mdl.sum(B[i][j]*nextc[i][j] for i in range
    (0,len(B)) for j in range(0,len(nextc[i]))) # In B vi
    sono i vettori b che contengono i tempi di attesa ( con
     il 40- davanti) moltiplicati per vettori priorita'.
    Bisogna massimizzare tutti i vettori b.

mdl.maximize(max_vars_func) # questa funzione calcolera' i
     valori degli scostamenti ottimi ed inoltre i valori
    contenuti in nextc, che indicano quale corsa della
    linea in cui si vuole salire e' la piu' adatta in base
    all'oraro di arrivo in uno stop.
```

La funzione di massimizzazione riceve i valori contenuti in B, moltiplicati per nextc. In output si avranno i valori calcolati per Scostamenti e per nextc.

Il resto del codice sono funzioni che servono per popolare Matrice, A, B e Intersezioni prima di chiamare la funzione di massimizzazione.
In seguito le altre figure del codice.

2. funzione iniziale

```
for linea in range(0,len(linee)): # per ogni linea
    for next_linea in range(0,len(linee)): # e per ogni linea successiva
        if(linea != next_linea): # se sono diverse
            si_intersecano=0  # settiamo si intersecano a 0
            for stop in range(0,Nstop): # per ogni stop
                if(linee[next_linea][stop]!=0 and linee[linea][stop]!=0): # se entrambe passano per quello stop
                    macro(linea,next_linea,stop,Matrice) # chiamiamo la funzione che popolera' le matrici
                    intersezione=intersezioni[linea,next_linea] #prendiamo l'oggetto contenuto in intersezionie[linea,next_linea]
                    if intersezione==0: #questo oggetto e' inizialmente settato a 0, quindi se risulta essere 0 si trasforma
                        intersezione=[] # in un vettore
                    intersezione.append(stop) # si aggiunge lo stop in cui linea e next_linea si intersecano
                    intersezioni[linea,next_linea]=intersezione # poi si inserisce nuovamente nella matrice alla riga linea e alla colonna next_linea
                    si_intersecano=1 # si setta si_intersecano a 1
            if si_intersecano==0: # se dopo il for si_intersecano non e' stato settato
                Matrice[linea,next_linea]=-1 # si inserisce dentro la matrice M il valore -1
                intersezioni[linea,next_linea]=-1 #si inserisce dentro la matrice delle intersezioni il valore -1
            else: # se e' stato settato si stampano le informazioni
                print("---la linea "+ str(linea)+" e la linea "+str(next_linea)+" si intersecano in :")
                if intersezioni[linea,next_linea]!=0:
                    for i in range(0,len(intersezioni[linea,next_linea])):
                        print("---stop: "+str(intersezioni[linea,next_linea][i]))

```

3. funzione macro

```
def macro(linea,next_linea,stop,Matrice): # riceve la linea, la next_linea , lo stop e la Matrice
    pos=stop_pos[stop][next_linea] # si ottiene la riga corrispondente a quella linea per lo stop specificato
    vet=zeros(corse_per_linea[linea], dtype=object)
    mat=Matrice[linea,next_linea]

    for i in range(0,corse_per_linea[linea]): #per tutte le corse della linea
        orario_arrivo=OrarioArrivo(stop,linea,i) #si calcola un ipotetico orario di arrivo per quella corsa i appartenente a linea, nello stop specificato
        idx=Tempo(stop,next_linea,orario_arrivo,linea,i) # questa funzione calcola il tempo di attesa e ritorna un indice
        vet[i]=idx # l'indice viene inserito dentro il vettore precedentemente creato
    if mat==0: # se l'oggetto preso da Matrice[linea, next_linea] e' ancora 0, quindi non contiene nulla
        mat=[] # si rendo l'oggetto mat un vettore
    mat.append(vet) # si appendo il vettore vet
    Matrice[linea,next_linea]=mat # si setta mat in Matrice alla riga e colonna corrispondente
```

4. funzione OrarioArrivo

```
def OrarioArrivo(stop,linea,corsa): # riceve stop, linea e corsa
    pos_linea=stop_pos[stop][linea] # identifica la posizione della linea nello stop spoecificato
    oa=AllStop[stop][pos_linea,corsa]
    oa_i=(oa//100)*60
    oa_r=oa%100
    oa_f=oa_i+oa_r+scostamento_corsa(linea,corsa)
    orarioarrivo=oa_f# si aggiunge al tempo indicato in Stops uno scostamento calcolato nella funzione scostamento_corsa
    return orarioarrivo# ritorna l'orario di arrivo
```

5. funzione Scostamento corsa.

```
def scostamento_corsa(next_linea,corsa):
    # riceve la linea e la corsa appartenente alla linea
    return sum(T[k]*Scostamenti[next_linea][corsa,k] for k in range(0,Nscostamenti)) # calcola la sommatoria del prodotto fra i valori di T e i valori binari contenuti in Scostamenti[next_linea][corsa]
```

6. funzione Tempo

```
def Tempo(stop,next_linea,orario_arrivo,linea,corsa): # riceve lo stop, la next_linea, l'orario di arrivo per
    # la linea , la linea, e la corsa
    pos_linea=stop_pos[stop][next_linea] # si identifica la posizione di next_linea per stop specificato
    nextcorsa= mdl.binary_var_dict(corse_per_linea[next_linea]+1,name="y"+str(len(nextc))+"_"+str(linea)+"_"+str(next_linea)+"_stop_"+str(stop)+"_corsa_"+str(corsa))  #nextcorsa e' un vettore binario, che indica quale corsa appartenente a next_linea e' da considerare per ottenere un tempo di attesa minimo. Associa ad ogni corsa di next_linea il valore 0 o 1. La corsa che avra' 1 sara' quella che restituira' un tempo di attesa minimo. Questo vettore binario viene calcolato dalla funzione di massimizzazione.
    idx=len(nextc) # si ottiene l'indice idx corrispondente all'attuale lunghezza del vettore nextc, il quale contiene tutti vettori di tipo nextcorsa.
    nextc.append(nextcorsa) # si appende nextcorsa a nextc

    a=zeros(corse_per_linea[next_linea]+1,dtype=object) # questo e' un vettore contenente i tempi di attesa, serve per i vincoli
    b=zeros(corse_per_linea[next_linea]+1,dtype=object) # questo vettore verra' dato in pasto alla funzione di ottimizzazione e contiene i tempi di attesa moltiplicati per i rispettivi valori priorita'.

    #somma=mdl.sum((40-(Stops[stop][pos_linea,i]+scostamento_corsa(next_linea,i)-orario_arrivo))*nextc[idx][i] for i in range(0,3))

    for j in range(0,corse_per_linea[next_linea]+1): # per quante sono le corse della next_linea
        op=AllStop[stop][pos_linea,j]
        op_i=(op//100)*60
        op_r=op%100
        op_f=op_i+op_r+scostamento_corsa(next_linea,j)

        a[j]=40-(op_f - orario_arrivo) # si popola il vettore a inserendo l'ipotetico tempo di attesa, orariopartenza-orarioarrivo, considerando la corsa j di next_linea e il relativo scostamento. Si impone 40 - per renderlo un problema di massimizzazione
        b[j]=stop_priority[stop]*a[j] # si popola il vettore b , inserendo il valore di a * i valori rispettivi di priorita'.

```

```
22      A.append(a) # si popola quindi la matrice A che
        contiene tanti vettori a, i quali contengono i tempi di
         attesa
23
24      B.append(b) # si popola La matrice B che contiene
        tanti vettori b, i quali contengono i tempi di attesa*
        priorita'
25
26
27      # l'indice in cui viene aggiunto a in A , b in B e
        nextcorsa in nextc e' lo stesso, ed e' indicato da idx.
         Il valore espresso da idx serve in quanto viene
        inserito in Matrice. Quando verranno calcolati gli
        scostamenti ( quindi verra' popolato il vettore di
        matrici Scostamenti ) e tutti i vettori nextcorsa, il
        valore in idx indichera' quale corsa della linea in cui
         si vuole salire e' da considerare, dato uno stop e un
        orario di arrivo.
28
29      return idx
```

Capitolo 4

Quantum computing e Big Data

4.1 Impostazione del problema per il quantum

I computer quantistici oggi disponibili sono chiamati macchine quantistiche a breve termine poiché non forniscono un'esecuzione priva di errori, anche con l'uso di milioni di qubit. Hanno risorse limitate e per questo sono molto popolari gli algoritmi di ottimizzazione ibridi che consistono nell'usare il calcolo sia classico che quantistico.
Come spiegato nel paragrafo 2.3.2 vi sono algoritmi correlati alla ricerca di stati fondamentali di un hamiltoniano (VQE, QAOA) e l'algoritmo di Grover che invece non utilizza la matrice Hamiltoniana di Ising.

4.1.1 Hamiltoniana di Ising

Il modello di Ising considera variabili discrete (con valori possibili pari a 1 o -1) chiamate spin.
Hamiltoniana è una funzione energetica che descrive l'energia del sistema. Nel modello classico di Ising con N spins con valori $s_i = \{-1,1\}$, gli hamiltoniani hanno la seguente forma:

$$H(s_1, ..., s_N) = -\sum_{i<j} J_{i,j} s_i s_j - \sum_{i=1}^{N} h_i s_i \tag{4.1}$$

Ji, j rappresenta la forza di interazione tra gli spin si e sj mentre hi la forza esterna che interagisce sullo spin si. La versione quantistica dell'Hamiltoniano classico (chiamato anche Operatore Hamiltoniano) rappresenta l'energia del sistema quantistico ed è la somma di tutte le energie cinetiche e potenziali del sistema. I suoi autovalori rappresentano i valori energetici che questo sistema può avere. L'Hamiltoniana di Ising è costruita come somma degli operatori di Pauli σx, σy,

σz.
La versione quantistica di Ising Hamiltonian ha la seguente forma:

$$H(\sigma_z^1, ..., \sigma_z^n) = -\sum_{i<j} J_{i,j}\sigma_z^i\sigma_z^j - \sum_{i=1}^{N} h_i\sigma_x^i \tag{4.2}$$

dove Ji, j rappresenta la forza di interazione tra gli spin σi e σj mentre hi la forza esterna che interagisce sullo spin σi.

L'algoritmo VQE, come QAOA, approssima i valori di una matrice H, che nella maggior parte dei casi è l'hamiltoniana del sistema. Il concetto di base è che la l'autovalore minimo della matrice viene utilizzato per approssimare l'energia dello stato fondamentale del sistema, cioè lo stato in cui il sistema ha l'energia più bassa. VQE è composto da una parte classica e una quantistica.

Prendendo in considerazione l'algoritmo VQE, viene spiegato come avviene la ricerca dello stato fondamentale minimo.
Essa consiste in vari step (Figura 4.1):

1. Preparazione dello stato quantistico, che verrà elaborato su moduli quantistici.

2. Calcolo dei valori attesi per lo stato preparato.
   Lo stato quantistico viene passato ai moduli quantistici che calcolano H_i.

3. I risultati dei moduli quantistici vengono passati alla CPU che calcola la somma dei H_i e quindi secondo il principio variazionale vengono calcolati gli autovalori. Il principio variazionale si basa sul presupposto che il più piccolo autovalore di H_{is} sia sempre minore del valore atteso.

4. l' algoritmo classico di ottimizzazione viene eseguito sulla CPU e crea un nuovo set di parametri. Il risultato dell'ottimizzatore classico viene passato all'unità di elaborazione Quantum.

   I passaggi 1-4 vengono ripetuti finché non viene raggiunto il minimo

4.1.2 Passaggi per risolvere il problema su un computer quantistico

Passaggi da eseguire per risolvere il problema:

- Si rappresenta il problema come hamiltoniano di Ising quantistico, in modo che lo stato fondamentale di H sia la soluzione alla domanda di ottimizzazione.

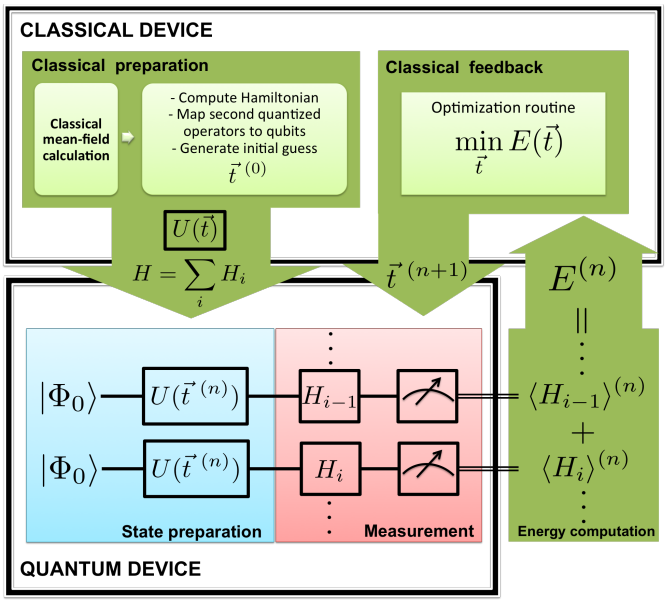


Figura 4.1. Funzionamento algoritmo VQE. Immagine tratta da *Strategies for quantum computing molecular energies using the unitary coupled cluster ansatz*, Romero J, Babbush R, McClean J, Hempel C, Love P, Aspuru-Guzik A, 2017

- Si sceglie l'algoritmo con cui si vuole risolvere il problema e i suoi parametri.
- Si sceglie il backend su cui verrà eseguito l'algoritmo. Può essere un simulatore o un vero computer quantistico.
- Si sceglie il numero di shots. Questo numero rappresenta quante ripetizioni del circuita bisogna eseguire. Questo parametro viene utilizzato per il campionamento.
- Si esegue il problema sul backend scelto in precedenza. Per ottenere lo stesso risultato ogni volta è necessario impostare il parametro seed.

4.1.3 DOcplex

Qiskit Aqua fornisce il modulo DOcplex [23] che genera l'Hamiltoniana di Ising per il problema di ottimizzazione specifico. Creare questo operatore da soli, senza alcuna libreria, è un processo molto complesso, quindi la possibilità di generarlo automaticamente è molto utile.
Per tradurre il problema in un problema risolvibile dal computer quantistico e generare l'Hamiltoniana di Ising, si deve prima creare il modello [24]. La creazione del modello consiste nello specificare il numero di variabili che verranno mappate sui qubit, specificare la funzione obiettivo e aggiungere vincoli.

Inoltre, alcune restrizioni devono essere soddisfatte per eseguire il DOcplex translate:

- Solo le variabili binarie possono essere utilizzate per rappresentare il problema
- I vincoli impostati possono essere vincoli di uguaglianza o disuguaglianza.
- La funzione che sarà ottimizzata (minimizzata o maximalizzata) deve essere lineare o quadratica.

Il problema trattato nel terzo capitolo è stato impostato in modo da rispettare le restrizioni descritte e poter generare automaticamente un Hemiltoniana. Nel prossimo paragrafo vengono specificati gli algoritmi con cui il problema è stato risolto, e l'approccio utilizzato.

4.2 Confronto dei risultati ottenuti

4.2.1 Approccio utilizzato vs brute force

E' stato implementata una soluzione cercando di calcolare la massimizzazione sul minor numero di dati possibili.
Coniderando il caso preso in analisi con 3 linee, 2 corse, 3 scostamenti e 2 stop, la funzione di massimizzazione lavora sulla matrice B che in questo caso avrà 72 celle, 24 righe e 3 colonne. Ogni cella rappresenta una possibile combinazione tra le corse delle linee, quindi la matrice B considera tutte le possibili combinazioni.
Ogni cella considera due corse di due linee diverse, ed ogni corsa ha 3 possibili scostamenti. Calcolare il valore di una cella di B significa calcolare il valore di 6 variabili binarie, 3 per corsa.
Calcolare il valore di ogni riga di B significa non solo calcolare il valore di 6 variabili binarie per cella, ma anche ulteriori 3 variabili binarie, una variabile per ogni colonna di ciascuna riga.
In totale avremo una dimensione della matrice B abbastanza ridotta considerando tutte le possibili combinazioni.
Il numero di variabili binarie totali calcolate è la somma di 27 (Dimensione della struttura dati Scostamenti, 3 corse per tre linee per tre scostamenti) e 72 (Dimensione della matrice B), quindi 99.

Se venisse utilizzato un approccio brute force si avrebbe un numero elevato di combinazioni. Si consideri la versione base del problema

- 3 linee
- 2 corse + la corsa del giorno dopo

- 3 scostamenti
- 2 stop

Si consideri di creare una matrice per ogni possibile combinazione degli scostamenti delle corse, per poi scegliere quella che ci dà un risultato ottimo.

Si calcola in questo modo il numero di possibili combinazioni.
Si terrà in considerazione solo 1 stop in cui le tre linee si incontrano, per semplificare.
Per ogni linea si avrebbero 9 combinazioni (3 scostamenti su 2 corse).

Di seguito le possibili combinazioni della Linea1.

Combinazione1	1000+0	1020+0	3400+0
Combinazione2	1000+2	1020+0	3400+2
Combinazione3	1000+0	1020+2	3400+0
Combinazione4	1000+2	1020+2	3400+2
Combinazione5	1000+0	1020+3	3400+0
Combinazione6	1000+3	1020+0	3400+3
Combinazione7	1000+3	1020+3	3400+3
Combinazione8	1000+2	1020+3	3400+2
Combinazione9	1000+3	1020+2	3400+3

In una matrice con 3 linee si avranno 729 combinazioni (9x9x9), quindi si otterranno 729 matrici.

Linea1	Combinazione1
Linea2	Combinazione1
Linea3	Combinazione1

Linea1	Combinazione1
Linea2	Combinazione1
Linea3	Combinazione2

Linea1	...
Linea2	...
Linea3	...

Linea1	Combinazione9
Linea2	Combinazione9
Linea3	Combinazione9

In ogni matrice ottenuta, poi, si dovranno generare tutte le possibili combinazioni fra le corse di due linee diverse in modo da calcolare il tempo di attesa minimo. Tenendo presente che ogni corsa della linea1, ad esempio, può essere accoppiata ad una corsa della linea2 e una corsa della linea3, si avranno 9 combinazioni possibili.

Per la prima corsa di ogni linea di ogni matrice.

Combinazione1	Linea1-corsa1, Linea2-corsa1, Linea3-corsa1
Combinazione2	Linea1-corsa1, Linea2-corsa1, Linea3-corsa2
Combinazione3	Linea1-corsa1, Linea2-corsa1, Linea3-corsa3
Combinazione4	Linea1-corsa1, Linea2-corsa2, Linea3-corsa1
Combinazione5	Linea1-corsa1, Linea2-corsa2, Linea3-corsa2
Combinazione6	Linea1-corsa1, Linea2-corsa2, Linea3-corsa3
Combinazione7	Linea1-corsa1, Linea2-corsa3, Linea3-corsa1
Combinazione8	Linea1-corsa1, Linea2-corsa3, Linea3-corsa2
Combinazione9	Linea1-corsa1, Linea2-corsa3, Linea3-corsa3

Quindi per ogni corsa di ogni linea avremo nove possibili tempi di attesa; per ogni linea quindi si avranno 81 possibili combinazioni di tempi di attesa (9x9) dato che viene calcolato il tempo di attesa per due corse in ogni linea (la prima e la seconda).

Per ogni linea di ogni matrice si avrà:

corsa1, combinazione1	corsa2, combinazione1
corsa1, combinazione1	corsa2, combinazione2
corsa1, combinazione1	corsa2, combinazione3
corsa1, ...	corsa2, ...
corsa1, combinazione9	corsa2, combinazione9

In totale per la matrice si avranno 531441 (81x81x81) possibili tempi di attesa, 81 per ogni linea , 9 per ogni corsa.
Per concludere, considerando tutte le matrici precedentemente ottenute, si avrebbero $81^3 \times 9^3$ circa 387 mila possibili combinazioni di tempi di attesa.

Delle 729 matrici generate solo una ha la combinazione vincente degli scostamenti ideali, mentre delle oltre 387 mila combinazioni di tempi di attesa solo tre sono le vincenti, con valori di tempi di attesa minimi, cioè una combinazione ottima per linea (3) ed appartenente alla matrice vincente.
In questo caso è stato considerato solo 1 stop per semplicità.
Considerando entrambi gli stop, il problema nella sua completezza avrebbe 282.429.536.481

possibili combinazioni di tempi di attesa, 729 matrici di scostamenti, in totale, quindi, 205.891.132.094.649 possibili combinazioni totali.
Si otterrebbe così un numero sproporzionato di matrici e combinazioni da iterare.

Va da sè che l'approccio utilizzato rispetto a quello appena descritto è più ottimale, ed è stato possibile utilizzarlo grazie all'uso delle variabili binarie, che possono assumere il valore 0 o 1 in base alla funzione obiettivo e ai vincoli.
Tramite l'uso di queste variabili e l'implementazione del modello con Docplex, si avrà una sola matrice (24 x 3) che la funzione maximize ottimizzerà, svolgendo tutti gli opportuni calcoli in maniera autonoma.
Importante quindi risulta essere la dimensione della matrice data in pasto alla funzione di ottimizzazione, affinché i calcoli avvengano in maniera più o meno rapida.

Di seguito viene illustrata la risoluzione del problema tramite l'ottimizzatore di default Cplex. Vengono alterati i dati in input e vengono calcolati i tempi di risoluzione.

4.2.2 CPLEX

Il problema risulta risolvibile senza difficoltà utilizzando l'ottimizzatore di default CPLEX.

Di seguito viene riportata la tabella in cui a partire dai dati specificati precedentemente, vengono generati differenti modelli alterando il numero di linee, corse, scostamenti e stop.

Si altera il numero di scostamenti, fissando il numero di linee (3) , il numero di corse (2) e il numero di stop (2).

N. Scost.	Matrice	Comb.Scost.	Comb.Tempi	Prova 1	Prova 2	Prova 3
2	24×3	64	$\simeq 282 \times 10^9$	$\simeq$ 20 s	$\simeq$ 21 s	$\simeq$ 18 s
3	24×3	729	$\simeq 282 \times 10^9$	$\simeq$ 46 s	$\simeq$ 1m 10s	$\simeq$ 54 s
4	24×3	4096	$\simeq 282 \times 10^9$	$\simeq$ 4m 3s	$\simeq$ 5m 50s	$\simeq$ 3m 40s
5	24×3	15625	$\simeq 282 \times 10^9$	$\simeq$ 1h 15m	$\simeq$ 1h 20m	$\simeq$ 1h 10m

Si può notare dalla tabella in alto che il numero di scostamenti incide solo sul valore delle combinazioni nella terza colonna, cioè le combinazioni dovute agli scostamenti.
Le combinazioni espresse nella quarta colonna, che sono dovute al calcolo della corsa ottima da prendere e quindi al calcolo del tempo di attesa, sono indipendenti dal numero di scostamenti.
In particolare, nella terza colonna il valore è ottenuto dal seguente calcolo:
Si consideri 2 scostamenti, per ogni linea $2^2 = 4$ combinazioni, cioè 2 scostamenti su 2 corse , in una matrice con 3 linee si avranno quindi $4^3 = 64$ combinazioni.

Le combinazioni espresse invece nella quarta colonna dipendono dalla dimensione della matrice A e B introdotte nel terzo capitolo. Queste matrici contengono tutte le possibili combinazioni delle corse negli stop considerati.
Il numero di righe indica quante interconnessioni ci possono essere in totale, il numero di colonne invece dipende dal numero di corse della linea che si desidera prendere.
Nelle celle di una stessa riga sono presenti i valori dei tempi di attesa, considerando lo stesso tempo di arrivo ma variando il tempo di partenza che dipende dalla corsa della linea successiva.
Quindi, se la linea che un utente vuole prendere ha 3 corse, la matrice in quella riga avrà tre colonne, in ognuna della quali si considera una corsa differente .

Infine, i valori della quarta colonna sono ottenuti dal seguente calcolo: considerando la dimensione della matrice 24x3 si avranno 3^{24} possibili combinazioni, cioè 282×10^9.

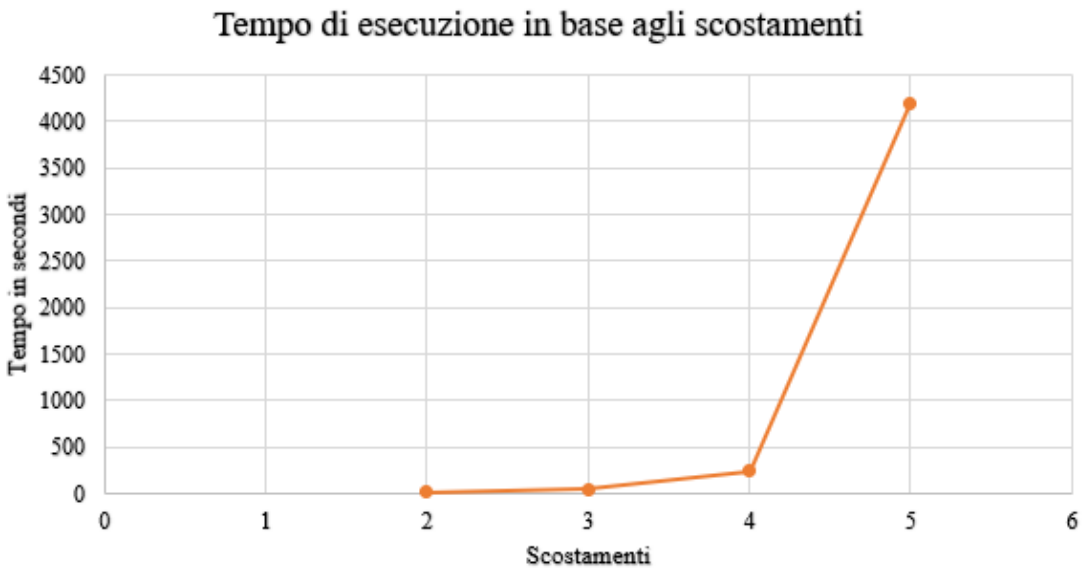


Figura 4.2. Tempo di esecuzione in base agli scostamenti

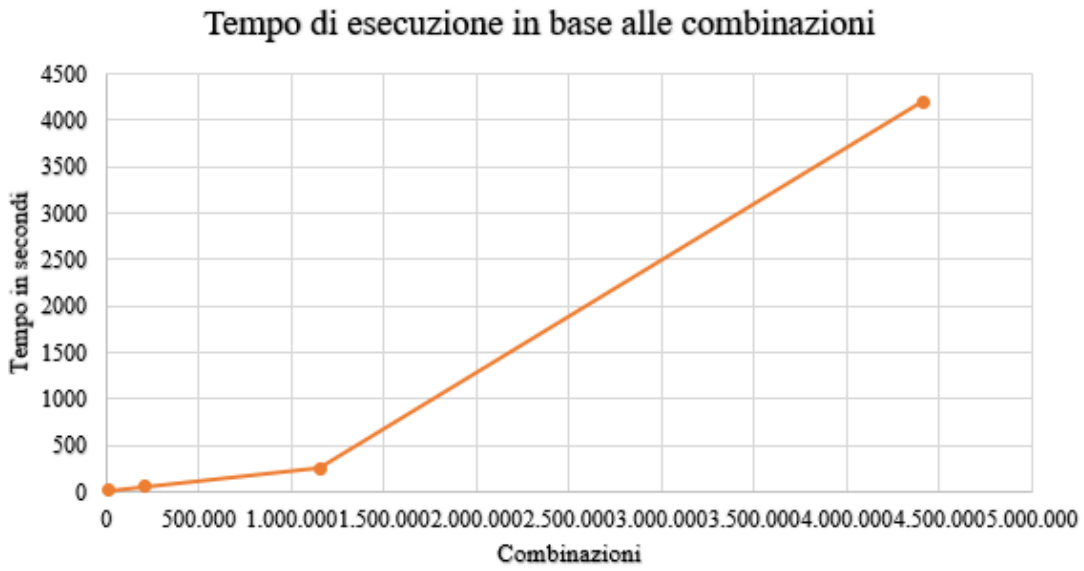


Figura 4.3. Tempo di esecuzione in base alle combinazioni totali

Si altera il numero di linee, fissando il numero di scostamenti (3) , il numero di corse (2) e il numero di stop (2).

N. Linee.	Matrice	Comb.Scost	Comb.Tempi	Prova 1	Prova 2	Prova 3
2	8×3	81	$\simeq 6.561$	$\simeq$ 15 s	$\simeq$ 18 s	$\simeq$ 20 s
3	24×3	729	$\simeq 282 \times 10^9$	$\simeq$ 46 s	$\simeq$ 1m 10s	$\simeq$ 54 s
4	48×3	6561	$\simeq 79.766 \times 10^{18}$	$\simeq$ 6m 21s	$\simeq$ 6m 50s	$\simeq$ 7m 13s
5	80×3	59.049	$\simeq 1{,}478e^{38}$	+2h	+2h	+2h

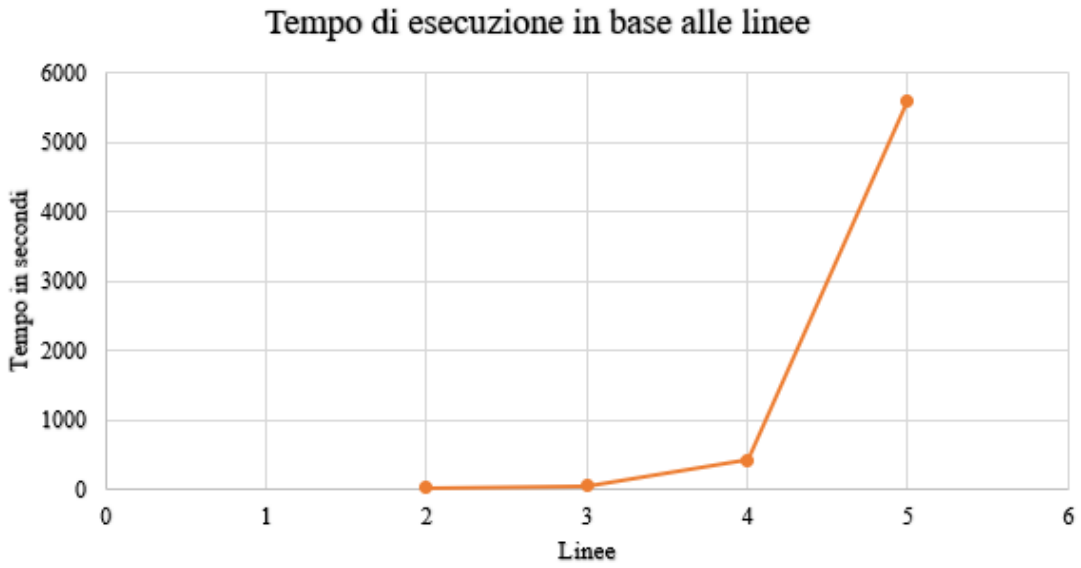


Figura 4.4. Tempo di esecuzione in base alle linee

Questo grafico è molto simile al 4.2.
Il numero di linee incide sulle combinazioni degli scostamenti, tuttavia anche sulle combinazioni dei tempi, a differenza del caso precedente.
Si può notare che i tempi di esecuzione si allungano in maniera significativa quando il numero di combinazioni degli scostamenti aumenta.

Considerando il caso successivo si avrà maggiore conferma di quanto appena dichiarato.

Si altera il numero di stop, fissando il numero di scostamenti (3) , il numero di linee (3) e il numero di corse (2).

N. Stop	Matrice	Comb.Scost	Comb.Tempi	Prova 1	Prova 2	Prova 3
2	24×3	729	$\simeq 282 \times 10^9$	46s	1m 10s	54s
3	36×3	729	$\simeq 150.094 \times 10^{12}$	56s	58s	1m 20s
4	48×3	729	$\simeq 79.766 \times 10^{18}$	1m 14s	2m 5s	1m 26s
5	60×3	729	$\simeq 42 \times 10^{27}$	2m 6s	2m 36s	3m 3s

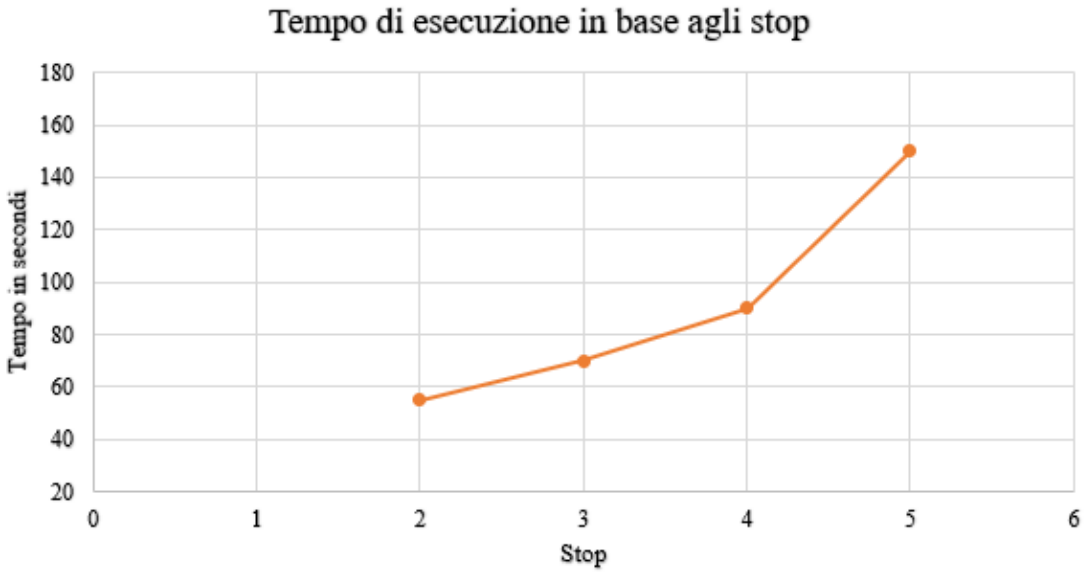


Figura 4.5. Tempo di esecuzione in base agli stop

In questo caso il numero di stop non incide sui valori della terza colonna, ma solo su quelli della quarta.
Aumentando infatti il numero di stop, aumentano le dimensioni della matrice A e B, ma non aumentano le combinazioni degli scostamenti.
Analizzando la tabella, si comprende che il numero di combinazioni dei tempi è aumentato in maniera esponenziale, passando da 10^9 a e^34, tuttavia i tempi di esecuzione sono cambiati poco. Come è stato accennato precedentemente, quindi, il dato che incide di più sulle prestazioni riguarda le combinazioni degli scostamenti, quindi il numero di corse e scostamenti.

Cambiando il numero di corse i tempi aumentano sproporzionatamente. Con 3 corse sono necessarie più di 2 ore. Non vengono quindi riportati i tempi di esecuzione siccome troppo lunghi, ma solo il numero di combinazioni.

N. Corse	Matrice	Comb.Scost	Comb.Tempi
2	24×3	729	$\simeq 282 \times 10^9$
3	24×4	19.683	$\simeq 281.474 \times 10^9$
4	24×5	531.441	$\simeq 59.604 \times 10^{12}$
5	24×6	14.706.125	$\simeq 59.604 \times 10^{12}$

4.2.3 VQE e QAOA

VQE e QAOA sono spesso menzionati insieme.
VQE [25] è spesso descritto come un algoritmo valido per i calcoli chimici e QAOA per l'ottimizzazione combinatoria.
Vale la pena capire la relazione tra i due.

Questi due algoritmi sono in grado di risolvere un problema QUBO trovando lo stato fondamentale di una corrispondente Hamiltoniana di Ising. Per stato fondamentale si intende lo stato in cui il sistema ha l'energia più bassa, ed è rappresentato dall'autovalore minimo.
Di seguito i maggiori dettagli.

Data una matrice hermitiana H con un autovalore minimo sconosciuto λmin, associato all'autostato $|\psi$min$\rangle$, VQE fornisce una stima $\lambda\theta$ che limita λmin:

$$\lambda_{min} \leq \lambda_\theta \equiv \langle\psi(\theta)|H|\psi(\theta)\rangle \tag{4.3}$$

dove $|\psi(\theta)\rangle$ è l'autostato associato a $\lambda\theta$. Applicando un circuito parametrizzato, rappresentato da $U(\theta)$, a uno stato iniziale arbitrario $|\psi\rangle$, l'algoritmo ottiene una stima $U(\theta)|\psi\rangle \equiv |\psi(\theta)\rangle$ su $|\psi\,\mathrm{min}\rangle$. La stima è ottimizzata iterativamente da un controller classico che modifica il parametro θ minimizzando il valore atteso di $\langle\psi(\theta)|H|\psi(\theta)\rangle$.

VQE è un'applicazione del metodo variazionale della meccanica quantistica. Il metodo variazionale è un metodo che si usa per ottenere una stima dell'energia dello stato fondamentale di un sistema.
Secondo questo metodo , un autovettore, $|\psi i\rangle$, di una matrice A è invariante sotto trasformazione di A fino ad una costante moltiplicativa scalare λi.

$$A|\psi_i\rangle = \lambda_i|\psi_i\rangle \tag{4.4}$$

Le matrici Hamiltoniane sono Hermitiane.
Il teorema spettrale afferma che gli autovalori di una matrice Hermitiana devono essere reali. Poiché qualsiasi quantità misurabile deve essere reale, le matrici hermitiane sono adatte per descrivere gli hamiltoniani dei sistemi quantistici. Inoltre, H può essere espresso come

$$H = \sum_{i=1}^{N} \lambda_i |\psi_i\rangle\langle\psi_i| \tag{4.5}$$

dove ogni λi è l'autovalore corrispondente all'autovettore $|\psi i\rangle$. Inoltre, il valore atteso di H su uno stato quantistico arbitrario $|\psi\rangle$ è dato da

$$\langle H\rangle_\psi \equiv \langle\psi|H|\psi\rangle \tag{4.6}$$

Sostituendo H con la sua rappresentazione come somma ponderata dei suoi autovettori

$$\langle H\rangle_\psi = \langle\psi|H|\psi\rangle = \langle\psi|\left(\sum_{i=1}^{N} \lambda_i |\psi_i\rangle\langle\psi_i|\right)|\psi\rangle = \sum_{i=1}^{N} \lambda_i \langle\psi|\psi_i\rangle\langle\psi_i|\psi\rangle = \sum_{i=1}^{N} \lambda_i |\langle\psi_i|\psi\rangle|^2 \tag{4.7}$$

L'ultima equazione dimostra che il valore atteso di un osservabile su qualsiasi stato può essere espresso come una combinazione lineare utilizzando gli autovalori associati ad H come pesi, quindi

$$\lambda_{min} \leq \langle H\rangle_\psi = \langle\psi|H|\psi\rangle = \sum_{i=1}^{N} \lambda_i |\langle\psi_i|\psi\rangle|^2 \tag{4.8}$$

L'equazione di cui sopra è nota come metodo variazionale.
È importante notare che questo implica che il valore atteso di qualsiasi funzione d'onda sarà sempre almeno l'autovalore minimo associato a H. Inoltre, il valore atteso dell'autostata $|\psi min\rangle$ è dato da $\langle\psi min|H|\psi min\rangle = \langle\psi min|\lambda min|\psi min\rangle = \lambda min$. Quindi, come previsto, $\langle H\rangle\psi min = \lambda min$.
Concludendo, quando l'Hamiltoniana di un sistema è descritta dalla matrice hermitiana H, l'energia dello stato fondamentale di quel sistema, Egs, è il più piccolo autovalore associato a H. Selezionando arbitrariamente una funzione d'onda $|\psi\rangle$ (chiamata ansatz) come ipotesi iniziale approssimativa $|\psi min\rangle$, calcolando il suo valore atteso, $\langle H\rangle\psi$, e aggiornando iterativamente la funzione d'onda, si possono ottenere limiti arbitrariamente stretti sull'energia dello stato fondamentale di un'Hamiltoniana.

Per implementare il metodo variazionale su un computer quantistico è necessario variare l'ansatz attraverso un approccio sistematico. VQE lo fa attraverso l'uso di un circuito parametrizzato con una forma fissa.

Un tale circuito è spesso chiamato forma variazionale e la sua azione può essere rappresentata dalla trasformazione lineare $U(\theta)$. Una forma variazionale viene applicata a uno stato iniziale $|\psi\rangle$ e genera uno stato di uscita $U(\theta)|\psi\rangle \equiv |\psi(\theta)\rangle$. L'ottimizzazione iterativa su $|\psi(\theta)\rangle$ mira a fornire un valore di aspettativa $\langle\psi(\theta)|H|\psi(\theta)\rangle \approx Egs \equiv \lambda min$. Idealmente $\psi(\theta)\rangle$ sarà vicino a $|\psi min\rangle$.

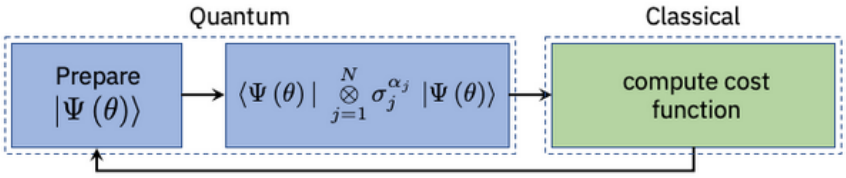


Figura 4.6. Esempio di algoritmo ibrido. Immagine tratta da qiskit.org

Una delle differenza tra VQE e QAOA è l'obiettivo dell'algoritmo.
In VQE si vuole trovare l'energia dello stato fondamentale e per farlo si deve riprodurre lo stato fondamentale. In QAOA invece l'obiettivo è trovare la soluzione al problema. Per fare ciò non c'è bisogno di trovare lo stato fondamentale, è necessario solo trovare uno stato che abbia una probabilità abbastanza alta di trovare la soluzione giusta.

Per applicare l'algoritmo QAOA al caso preso in analisi, si procede come illustrato.

```
qp = QuadraticProgram()
qp.from_docplex(model)
print(qp.export_as_lp_string())

provider = IBMQ.load_account()
backend = provider.get_backend("ibmq_qasm_simulator")

seed = 100
quantum_instance = QuantumInstance(backend, seed_simulator=seed,
    seed_transpiler=seed)
```

Si cre un modello docplex che definisce il problema preso in analisi, quindi si utilizza la funzione from_docplex () per convertire il modello in un QuadraticProgram, che poi verrà dato in pasto all'algoritmo scelto, in questo caso QAOA.

Viene instanziato il quantum_instance, specificando il parametro seed
e il simulatore scelto

Utilizzando come provider IBM vi sono vari simulatori disponibili.

Simulatore	Numero di qubits
ibmq_qasm_simulator	32
ibmq_16_melbourne	15
ibmq_ourense	5
ibmqx2	5
ibmq_vigo	5
ibmq_armonk	1

Utilizzando invece come provider AerProvider, si avranno i seguenti simulatori.

Simulatore	Numero di qubits
qasm_simulator	24
statevector_simulator	24
unitary_simulator	14

L'impostazione di seed su un dato valore garantirà la prevedibilità del risultato, serve per controllare il campionamento quando il backend è un simulatore.
Il simulatore più potente, al momento, è ibmq_qasm_simulator con 32 qubits disponibili.
Successivamente vengono impostati i parametri dell'algoritmo [26] ed eseguito.

```
spsa = SPSA(max_trials=250)
qaoa = QAOA(optimizer=spsa, p=5, quantum_instance=quantum_instance)
algorithm = MinimumEigenOptimizer(qaoa)
results = algorithm.solve(qp)
print("x={}".format(results.x))
print("fval={}".format(results.fval))
```

- L'implementazione QAOA in Aqua estende direttamente VQE ed eredita la struttura generale di ottimizzazione ibrida di VQE. Tuttavia, a differenza di VQE, che può essere configurato con forme variazionali arbitrarie, QAOA utilizza la propria forma variazionale ottimizzata, che dipende dal valore di un numero intero positivo p.
  P determina la profondità della forma variazionale, e quindi influenza la qualità dell'approssimazione.
  In particolare l'algoritmo dipende da p e la qualità dell'approssimazione migliora all'aumentare di p. P ha un valore minimo valido di 1.

- SPSA [27], approssimazione stocastica della perturbazione simultanea, è l' ottimizzatore classico.
  E' un metodo in grado di trovare minimi globali, condividendo questa proprietà con altri metodi come l'annealing simulato.

Questo ottimizzatore è ideale per diversi motivi.
La sua caratteristica principale è l'approssimazione del gradiente, che richiede solo due misurazioni della funzione obiettivo, indipendentemente dalla dimensione del problema di ottimizzazione.
Ciò elimina la necessità di effettuare più misurazioni del gradiente dello stato quantistico (che sono spesso difficili o impossibili da ottenere).
In secondo luogo, SPSA richiede un numero relativamente inferiore di misurazioni per trovare una soluzione adeguata alla funzione obiettivo.
Infine, SPSA è in grado di accogliere il rumore.
Può essere utilizzato in presenza di rumore ed è quindi indicato in situazioni che implicano incertezza di misura su un calcolo quantistico quando si trova un minimo, ed è già necessario un numero significativo di misurazioni per ottenere una distinta distribuzione probabilistica. Se si sta eseguendo un algoritmo variazionale utilizzando un simulatore Quantum ASseMbly Language (QASM) o un dispositivo reale, SPSA sarebbe la scelta più consigliata tra gli ottimizzatori disponibili.

- Infine, come spiegato nel secondo capitolo, gli algoritmi VQE e QAOA calcolano gli stati fondamentali di un Hamiltoniano di Ising.
  Una classe di problemi che può essere risolta applicando questi due algoritmi sono i problemi di ottimizzazione binaria quadratica non vincolata (QUBO), in quanto trovare la soluzione a un QUBO equivale a trovare lo stato fondamentale di un corrispondente Hamiltoniano di Ising.
  Qiskit fornisce la conversione automatica da un QuadraticProgram adatto a un Hamiltoniano di Ising, e ciò consente di sfruttare i MinimumEigenSolver come VQE, QAOA o NumpyMinimumEigensolver (metodo esatto classico).
  MinimumEigenOptimizer converte un dato problema nella classe del problema supportata.
  Tradurrà, quindi, variabili intere in variabili binarie o aggiungerà vincoli di uguaglianza lineare come termine di penalità quadratica all'obiettivo.
  Il QUBO risultante viene tradotto in un'Hamiltoniana di Ising il cui autovettore minimo e il corrispondente autostato corrispondono alla soluzione ottima del problema di ottimizzazione originale.
  L'autostato minimo fornito viene utilizzato per approssimare lo stato fondamentale dell'Hamiltoniano per trovare una buona soluzione per il problema di ottimizzazione.

Purtroppo il problema preso in analisi non è risolvibile applicando QAOA. Come specificato il simulatore più potente ha 32 qubits, non sufficienti per trovare una soluzione al problema. Per poterlo risolvere bisgnerebbe ridurre il numero di dati in input, riducendo ad esempio il numero di stop o corse.

4.2.4 GROVER

Uno dei più grandi vantaggi del computer quantistico, è la sua velocità nelle ricerche nei database. L'algoritmo di Grover [10] può essere utilizzato per elevare al quadrato la velocità di ricerca in un database. In realtà, può essere anche utilizzato per eseguire molte altre operazioni alla stessa velocità, ma appunto l'utilizzo più comune risulta essere quello della ricerca in un database.

Si supponga di avere un lungo elenco di N elementi. Tra questi elementi c'è un oggetto con una proprietà unica che si desidera individuare; si chiamerà questo il vincitore w.

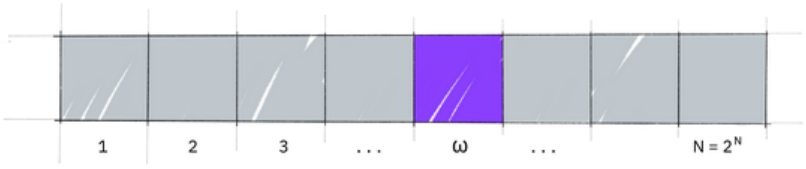


Figura 4.7. Elenco di N elementi. Immagine tratta da qiskit.org

Per trovare l'elemento contrassegnato utilizzando il calcolo classico, si dovrebbe controllare in media $N/2$ di queste caselle e, nel peggiore dei casi, tutte N di esse. Su un computer quantistico, tuttavia, si può trovare l'elemento contrassegnato in circa $\sqrt{N}$ passaggi con il trucco di amplificazione dell'ampiezza di Grover.
Inanzi tutto, dobbiamo codificare la lista in qubit, si sceglie una codifica per ogni elemento x, $w \in \{0,1\}^n$ con $N = 2^n$.

L'algoritmo di Grover risolve matrici che aggiungono una fase negativa agli stati della soluzione. Cioè per ogni stato $|x\rangle$ nella base computazionale:

$$U_\omega|x\rangle = \begin{cases} |x\rangle, & if\ x \neq \omega \\ -|x\rangle, & if\ x = \omega \end{cases} \tag{4.9}$$

La matrice è chiamata Oracle ed è una matrice diagonale, dove la voce che corrisponde all'elemento contrassegnato avrà una fase negativa. Ad esempio, se si hanno tre qubit e $\omega = 101$, la matrice sarà:

$$U_\omega = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.10}$$

Dove nella sesta riga vi è $\omega = 101$.
La matrice Oracle può quindi essere descritto come:

$$U_\omega|x\rangle = (-1)^{f(x)}|x\rangle \tag{4.11}$$

Dove è definita la funzione f tale che f(x) = 1 se x è l'elemento cercato w, e f(x) = 0 se non lo è.

Questa matrice può essere ricavata per ogni elemento senza saperne la posizione.

Se si hanno n qubits in superposition, la probabilità di ottenere il qubit corretto misurando il sistema casualmente è di $1/2^n$. Qui entra in gioco l'Amplitude Amplification, che servirà ad aumentare la probabilità che il qubit misurato sia proprio quello cercato. Si parte da uno stato in cui tutti i qubit hanno la stessa Amplitude di $1/2^n$:

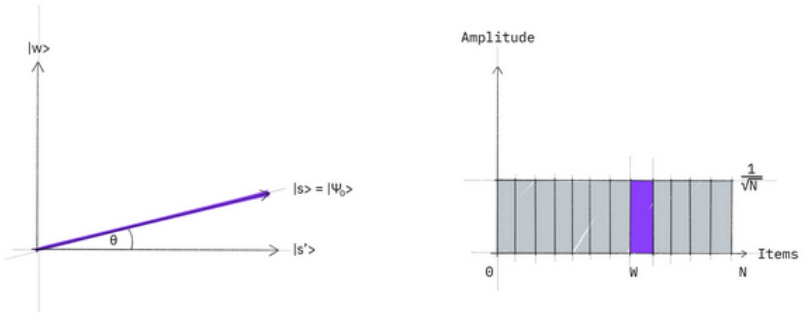


Figura 4.8. Stato iniziale dei qubits. Immagine tratta da qiskit.org

Viene applicata la matrice Oracle

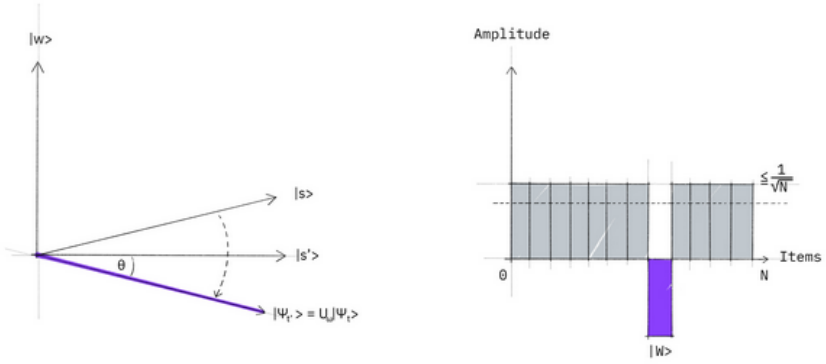


Figura 4.9. Applicazione della matrice Oracle. Immagine tratta da qiskit.org

Applichiamo ora un'ulteriore riflessione (Us) sullo stato $|s\rangle$: Us $= 2|s\rangle\langle s| - 1$. Questa trasformazione mappa lo stato su $UsUf|s\rangle$ e completa la trasformazione.

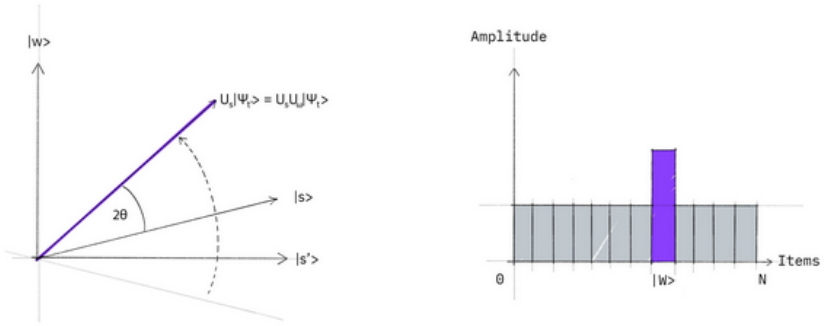


Figura 4.10. Applicazione della riflessione Us. Immagine tratta da qiskit.org

Due riflessioni corrispondono sempre a una rotazione. La trasformazione $UsUf$ ruota lo stato iniziale $|s\rangle$ più vicino al vincitore $|w\rangle$.
L'azione della riflessione Us nel diagramma a barre dell'ampiezza può essere intesa come una riflessione sull'ampiezza media. Poiché l'ampiezza media è stata abbassata dalla prima riflessione, questa trasformazione aumenta l'ampiezza negativa di $|w\rangle$ a circa tre volte il suo valore originale, mentre diminuisce le altre ampiezze. Si ritorna quindi alla seconda riflessione per ripetere l'applicazione.
Questa procedura verrà ripetuta più volte. Dopo t passi saremo nello stato $|\psi_t\rangle$ dove: $|\psi_t\rangle = (UsUf)^t|s\rangle$.

Si può vedere che l'ampiezza di $|w\rangle$ cresce linearmente con il numero di applicazioni $\sim tN^{-1/2}$, quindi sono sufficienti circa $\sqrt{N}$ rotazioni.

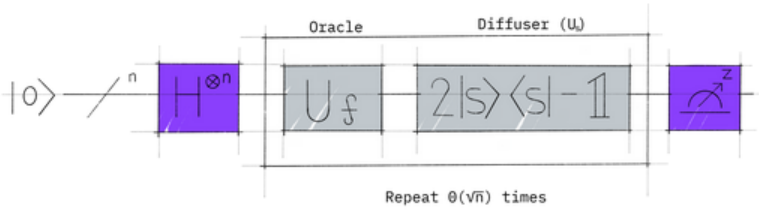


Figura 4.11. Circuito riassuntivo dell'algoritmo di Grover. Immagine tratta da qiskit.org

Per applicare l'algoritmo di Grover al problema preso in analisi si procede come segue.

```
qp = QuadraticProgram()
qp.from_docplex(mdl)
print(qp.export_as_lp_string())
provider = IBMQ.load_account()
backend = provider.get_backend("ibmq_qasm_simulator")
seed = 100
quantum_instance = QuantumInstance(backend, seed_simulator=seed,
    seed_transpiler=seed)
```

Vengono specficati nel paragrafo precedente i parametri presenti nel codice riportato.

```
grover_optimizer = GroverOptimizer(1, num_iterations=2,
    quantum_instance=quantum_instance)
results = grover_optimizer.solve(qp)
print("x={}".format(results.x))
print("fval={}".format(results.fval))
```

L'algoritmo di Grover, a differenza degli algoritmi analizzati nel capitolo precedente, non si basa sulla ricerca di stati fondamentali di un hamiltoniano.
I parametri da specificare nell'algoritmo sono [28]:

- num_value_qubits (int) - Il numero di qubit da usare, si ricorda il massimo è di 32 qubits .
- num_iterations (int) - Il numero di iterazioni che l'algoritmo cercherà senza alcun miglioramento .

Anche in questo caso, come per l'algoritmo QAOA, non è possibile risolvere il problema preso in analisi siccome non vi sono abbastanza qubits. Bisogna quindi ridurre il numero di dati in input.

4.3 Limiti del Quantum Computing

Come si è visto nei paragrafi precedenti, uno dei limiti attualmente presenti nei computer quantistici è il numero ancora limitato di qubits.

Inoltre i computer quantistici sono estremamente difficili da progettare, costruire e programmare. Di conseguenza, presentano da errori sotto forma di rumore, difetti e perdita di coerenza quantistica, che è fondamentale per il loro funzionamento.
La perdita di coerenza (chiamata decoerenza), causata da vibrazioni, fluttuazioni di temperatura, onde elettromagnetiche e altre interazioni con l'ambiente esterno, influenzano le prestazioni e la correttezza dei risultati del quantum computing [29].
Le tecnologie e le architetture concorrenti stanno affrontando questi problemi, tuttavia nessuna piattaforma hardware esistente può mantenere la coerenza e fornire la solida correzione degli errori richiesta per il calcolo su larga scala.
Inoltre, il tasso di errore fisico cresce con la crescita del computer e, aumentandone le dimensioni, gli schemi di tolleranza ai guasti non possono più ridurre l'errore di calcolo a un numero arbitrariamente piccolo. C'è un minimo errore computazionale raggiungibile, oltre il quale un'ulteriore crescita del computer nel tentativo di ridurre l'errore diventa controproducente.

Come si possono ottenere risultati utili da un computer che è insolitamente inaffidabile?

Le risposte provengono da ricercatori dell'industria, del mondo accademico e dei laboratori nazionali che perseguono una varietà di metodi per ridurre gli errori.

Un approccio consiste in algoritmi ibridi quantistico-classici che eseguono solo le sezioni più critiche per le prestazioni di un programma su un computer quantistico e la maggior parte del programma in esecuzione su un computer classico più robusto.
Queste e altre strategie si stanno dimostrando utili per affrontare l'ambiente rumoroso dei computer quantistici di oggi.

Sebbene anche i computer classici siano influenzati da varie fonti di errore, questi errori possono essere corretti con una modesta quantità di memoria e logica extra. Esistono schemi di correzione degli errori anche per i computer quantistici, tuttavia consumano un numero così elevato di qubit che rimangono relativamente pochi qubit per il calcolo effettivo. Ciò riduce significativamente le dimensioni di un problema a una frazione minuscola rispetto a ciò che potrebbe essere eseguito su hardware privo di difetti.

Capitolo 5

Conclusioni

Nel mio lavoro di tesi viene descritto il processo di manipolazione dei qubits attraverso i gates, confrontando questa metodologia, propria dei computer IBM, con quella dei computer D-Wave. Vengono analizzate le varie applicazioni del quantum e specificata la libreria usata (Qiskit-aqua).

Il problema preso in esame è un problema di ottimizzazione legato alle linee di trasporto. Si è scelto di prendere in analisi le linee di trasporto della città di Torino, in particolare le linee dei bus. Ogni linea ha un numero di corse giornaliere e durante il tragitto il bus si ferma in determinate fermate, ad un orario di arrivo prestabilito. Ogni corsa si interconnette con le altre linee di trasporto nelle varie fermate.
Il programma implementato deve calcolare gli spostamenti ottimali degli orari delle corse, affinché ci sia massima interconnessione fra esse. Per interconnessione si intende uno stop in cui due o più corse si interconnettono. Ogni stop (fermata) ha una determinata priorità. Calcolare la massima interconnessione significa calcolare lo scostamento ideale per ogni corsa, affinché ci sia un valore massimo legato alla priorità degli stop. Il problema, inoltre, deve tener conto del tempo di attesa, calcolato come differenza tra l'orario di partenza e l'orario di arrivo di due linee diverse, in un determinato stop. L'obiettivo è ottenere un valore massimo legato alle priorità, e anche un valore minimo legato ai tempi di attesa.

E' stata considerata una versione del problema molto semplice, in particolare sono state esaminate 3 linee di trasporto, ognuna con 2 corse giornaliere. Le linee si incontrano in 2 stop, e gli scostamenti possibili sono 3 (di 0, 2 o 3 minuti).

Il problema è risultato essere abbastanza complesso, anche per un computer quantistico. La sua impostazione richiede, infatti, numerose strutture dati e le combinazioni da considerare vanno dall'ordine di 10^4 all'ordine di 10^{34}. Capire come risolvere il problema attraverso vincoli e variabili binarie è stata la parte più difficile, nonché il fulcro dello studio svolto.

Attraverso l'ottimizzatore classico Cplex è stato possibile trovare una soluzione, senza particolari difficoltà.

Considerando i seguenti dati in input:

Stop1	Corsa1	Corsa2	Corsa3
Linea1	10:00	10:20	10:34
Linea2	10:17	10:24	10:27
Linea3	10:12	10:17	10:32

Stop2	Corsa1	Corsa2	Corsa3
Linea1	10:05	10:25	10:39
Linea2	10:19	10:26	10:29
Linea3	10:20	10:25	10:40

rappresentati dal seguente codice

```
Stop1 = array([[1000, 1020,  1034],
               [1017, 1024,  1027],
              [1012,   1017,1032]])

Stop2 = array([[1005, 1025,  1039],
               [1019, 1026,  1029],
              [1020,   1025,1040]])
```

e considerando che lo stop 1 ha priorità 2, e lo stop 2 ha priorità 5,

```
stop_priority=array([2,5])
```

l'output del programma è il seguente

```
solution : Model1
objective: 2996

- linea 1 e linea 2
--si intersecano
---scendendo dalla corsa 1 allo stop 1 per salire sulla linea 2
   bisogna attendere un minimo di :
---17.0 minuti, la corsa giornaliera da prendere e la 1

---scendendo dalla corsa 2 allo stop 1 per salire sulla linea 2
   bisogna attendere un minimo di :
---3.0 minuti, la corsa giornaliera da prendere e la 2

---scendendo dalla corsa 1 allo stop 2 per salire sulla linea 2
   bisogna attendere un minimo di :
---14.0 minuti, la corsa giornaliera da prendere e la 1

```

```
---scendendo dalla corsa 2 allo stop 2 per salire sulla linea 2
   bisogna attendere un minimo di :
---0.0 minuti, la corsa giornaliera da prendere e la 2

- linea 1 e linea 3
--si intersecano
---scendendo ...
```

Come si evince, il programma calcola il valore massimo legato alle priorità degli stop, e il valore minimo dei tempi di attesa.
Dato un orario di arrivo e uno stop, viene indicata quale corsa della linea successiva garantisce una minora attesa, specificando il tempo di attesa espresso in minuti.

Sono state analizzate le prestazioni, variando di volta in volta il numero di linee, corse, stop e scostamenti. Nel caso più semplice, ottenuto variando il numero di stop, il computer riesce a trovare una soluzione in pochi minuti, invece, variando il numero di corse, sono necessarie di un paio di ore.

I tempi di esecuzione nel caso più semplice sono i seguenti.

N. Stop	Matrice	Comb.Scost	Comb.Tempi	Prova 1	Prova 2	Prova 3
2	24×3	729	$\simeq 282 \times 10^9$	46s	1m 10s	54s
3	36×3	729	$\simeq 150.094 \times 10^{12}$	56s	58s	1m 20s
4	48×3	729	$\simeq 79.766 \times 10^{18}$	1m 14s	2m 5s	1m 26s
5	60×3	729	$\simeq 2{,}25e^{34}$	2m 6s	2m 36s	3m 3s

Per quanto riguarda gli algoritmi di ottimizzazione Grover e QAOA, il problema non trova una soluzione perché il numero di qubits disponibili (32), usando il simulatore attualmente più potente, non è sufficiente. Mi sono imbattuta quindi nei limiti hardware del quantum, che sono stati accentuati dal collegamento da remoto al computer IBM.

Per dare un'idea delle difficoltà hardware è sufficiente pensare che i più grandi processori quantistici di Ibm possiedono 65 qubit, Google dispone di un prototipo a 72 qubit; tuttavia, servirebbero milioni di qubit per eseguire calcoli pratici.
Ad oggi, nessun prototipo si è spinto oltre quello che sarebbe possibile fare con un normale computer.

Nonostante i limiti del computer quantistico, i più importanti attori nel mercato IT stanno investendo molto in questo ambito e tra il 2013 ed il 2030 si prospettano importanti evoluzioni. È nelle roadmap sia di Ibm sia di Google la possibilità di realizzare dispositivi da un milione di qubit, già entro il 2030.

Bibliografia

[1] Turing, A.M. *On Computable Numbers, with an Application to the Entscheidungsproblem*, Proceedings of the London Mathematical Society, 1937

[2] Benioff P. *Quantum Mechanical Hamiltonian Models of Turing Machines* Journal of Statistical Physics, FoL 29, No. 3, 1982

[3] Bennett CH. *Logical reversibility of computation* Ibm Journal of Research and Development, 1973

[4] Feynman R. *Simulating physics with computers* International Journal of Theoretical Physics volume 21, pages467–488, 1982

[5] Deutsch D. *Quantum theory, the Church-Turing principle and the universal quantum computer* Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences, 1985

[6] Shor P. *Algorithms for quantum computation: discrete logarithms and factoring* Proceedings 35th Annual Symposium on Foundations of Computer Science,1994

[7] D.S. Johnson. *A catalog of complexity classes* In J. van Leeuwen, editor, Handbookof Theoretical Computer Science, volume A, pages 67–161, Elsevier, 1990.

[8] E. Bernstein, U. Vazirani *Quantum complexity theory* , STOC '93: Proceedings of the twenty-fifth annual ACM symposium on Theory of Computing, 1993

[9] A. Barenco, C. H. Bennett, R. Cleve,D. P. DiVincenzo, N. Margolus, P. Shor,T. Sleator, J. Smolin, e H. Weinfurter *Elementary gates for quantum computation*, Phys.Rev. A52, 1995

[10] L. K. Grover *A fast quantum mechanical algorithm for database search*, Proceedings of the 28th Annual ACM Symposium on the Theory of Computing , 1996

[11] Qiskit: Learning quantum computation using qiskit, the case for quantum. https://qiskit.org/textbook/ch-states/case-for-quantum.html

[12] Microsoft. Qubits https://docs.microsoft.com/it-it/quantum/concepts/the-qubit

[13] A. Einstein, B. Podolsky, and N. Rosen *Can Quantum-Mechanical Description of Physical Reality Be Considered Complete?* Phys. Rev. 47, 777 , 1935

[14] Peres A. Daniel R. Terno *Quantum Information and Relativity Theory*, 2004

[15] Sejuti D. Top Applications Of Quantum Computing Everyone Should Know About, https://analyticsindiamag.com/top-applications-of-quantum-computing-everyone-should-know-about/, 20 Marzo 2020

[16] Jackson M. 6 Things Quantum Computers Will Be Incredibly Useful For https://singularityhub.com/2017/06/25/6-things-quantum-computers-will-be-incredibly-useful-for/ 25 Giugno 2017

[17] Taylor-Smith K. Using Quantum Computing to Tell the Weather https://www.azoquantum.com/Article.aspx?ArticleID=98 14 Dicembre 2018

[18] Fernández Lorenzo S. How may quantum computing affect Artificial Intelligence? https://www.quantaneo.com/How-may-quantum-computing-affect-Artificial-Intelligence_a391.html, 21 Gennaio 2020

[19] Gossett S. 8 Quantum Computing Applications & Examples https://builtin.com/hardware/quantum-computing-applications 18 Novembre 2019

[20] J. Kiefer and J. Wolfowitz, *Stochastic Estimation of the Maximum of a Regression Function*, Ann. Math. Statist, Volume 23, Number 3 , 1952

[21] Wang U. What is Quantum Annealing and how does it differ from Gate based Quantum Computers? https://quantumzeitgeist.com/what-is-quantum-annealing-and-how-does-it-differ-from-gate-based-quantum-computers/ 24 Ottobre 2020

[22] Woerner S. (6 Luglio 2020) , *A Walkthrough of Qiskit's New Optimization Module*, Qiskit https://medium.com/qiskit/towards-quantum-advantage-for-optimization-with-qiskit-9a564339ef26

[23] Docplex.
https://qiskit.org/documentation/stubs/qiskit.optimization.applications.ising.docplex.html

[24] Docplex. Model
http://ibmdecisionoptimization.github.io/docplex-doc/mp/docplex.mp.model.html

[25] Peruzzo A, McClean J, *A variational eigenvalue solver on a photonic quantum processor*, Nature communications 5 (2014): 4213.

[26] Qiskit, qiskit.aqua.algorithms.QAOA
https://qiskit.org/documentation/stubs/qiskit.aqua.algorithms.QAOA.html

[27] Spall, J. C.*Multivariate Stochastic Approximation Using a Simultaneous Perturbation Gradient Approximation*,EEE Transactions on Automatic Control, vol. 37(3), pp. 332–341, 1992

[28] Qiskit. qiskit.aqua.algorithms.Grover
https://qiskit.org/documentation/stubs/qiskit.aqua.algorithms.Grover.html

[29] Aaronson S.*The limits of quantum computers* Scientific American, 298(3):62–69, 2008.